To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

User's Manual

16

The revision list can be viewed directly by clicking the title page.

The revision list summarizes the locations of revisions and additions. Details should always be checked by referring to the relevant text.

H8/300H Series

Software Manual

Renesas 16-Bit Single-Chip Microcomputer
H8 Family/H8/300H Series

Renesas Electronics
www.renesas.com

Rev. 3.00 2004.12

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.

General Precautions on Handling of Product

1. Treatment of NC Pins

Note: Do not connect anything to the NC pins.
The NC (not connected) pins are either not connected to any of the internal circuitry or are used as test pins or to reduce noise. If something is connected to the NC pins, the operation of the LSI is not guaranteed.

2. Treatment of Unused Input Pins

Note: Fix all unused input pins to high or low level.
Generally, the input pins of CMOS products are high-impedance input pins. If unused pins are in their open states, intermediate levels are induced by noise in the vicinity, a pass-through current flows internally, and a malfunction may occur.

3. Processing before Initialization

Note: When power is first supplied, the product's state is undefined.
The states of internal circuits are undefined until full power is supplied throughout the chip and a low level is input on the reset pin. During the period where the states are undefined, the register settings and the output state of each pin are also undefined. Design your system so that it does not malfunction because of processing while it is in this undefined state. For those products which have a reset function, reset the LSI immediately after the power supply has been turned on.

4. Prohibition of Access to Undefined or Reserved Address

Note: Access to undefined or reserved addresses is prohibited.
The undefined or reserved addresses may be used to expand functions, or test registers may have been be allocated to these address. Do not access these registers; the system's operation is not guaranteed if they are accessed.

Preface

The H8/300H Series is built around a 32-bit H8/300H CPU core with sixteen 16-bit registers, a concise, optimized instruction set designed for high-speed operation, and a 16-Mbyte linear address space. For easy migration from the H8/300 Series, the instruction set is upward-compatible with the H8/300 Series at the object-code level. Programs coded in the high-level language C can be compiled to high-speed executable code.

This manual gives details of the H8/300H CPU instructions and can be used with all microcontrollers in the H8/300H Series.

For hardware details, refer to the relevant microcontroller hardware manual.

Main Revisions for this Edition

Item	Page	Revisions (See Manual for Details)
All	—	All references to Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names changed to Renesas Technology Corp. Designation for categories changed from “series” to “group”

Contents

Section 1 CPU

1.1 Overview

The H8/300H CPU is a high-speed central processing unit with an internal 32-bit architecture that is upward-compatible with the H8/300 CPU. The H8/300H CPU has sixteen 16-bit general registers, can address a 16-Mbyte linear address space, and is ideal for realtime control.

1.1.1 Features

The H8/300H CPU has the following features.

- Upward-compatible with H8/300 CPU
  — Can execute H8/300 object programs
- General-register architecture
  — Sixteen 16-bit general registers (also usable as sixteen 8-bit registers or eight 32-bit registers)
- Sixty-two basic instructions
  — 8/16/32-bit arithmetic and logic instructions
  — Multiply and divide instructions
  — Powerful bit-manipulation instructions
- Eight addressing modes
  — Register direct [Rn]
  — Register indirect [@ERn]
  — Register indirect with displacement [@(d:16,ERn) or @(d:24,ERn)]
  — Register indirect with post-increment or pre-decrement [@ERn+ or @–ERn]
  — Absolute address [@aa:8, @aa:16, or @aa:24]
  — Immediate [#xx:8, #xx:16, or #xx:32]
  — Program-counter relative [@(d:8,PC) or @(d:16,PC)]
  — Memory indirect [@@aa:8]
- 16-Mbyte address space
- High-speed operation
  — All frequently-used instructions execute in two to four states
  — Maximum clock frequency: 16 MHz
  — 8/16/32-bit register-register add/subtract: 125 ns
  — 8 × 8-bit register-register multiply: 875 ns

- — 16 ÷ 8-bit register-register divide: 875 ns
- — 16 × 16-bit register-register multiply: 1375 ns
- — 32 ÷ 16-bit register-register divide: 1375 ns

- Two CPU operating modes
  - — Normal mode
  - — Advanced mode
- Low-power mode
  - — Transition to power-down state by SLEEP instruction

1.1.2 Differences from H8/300 CPU

In comparison to the H8/300 CPU, the H8/300H CPU has the following enhancements.

- More general registers
  Eight 16-bit registers have been added.
- Expanded address space
  Normal mode supports the same 64-kbyte address space as the H8/300 CPU.
  Advanced mode supports a maximum 16-Mbyte address space.
- Enhanced addressing
  The addressing modes have been enhanced to make effective use of the 16-Mbyte address space.
- Enhanced instructions
  Signed multiply/divide instructions and other instructions have been added.

1.2 CPU Operating Modes

The H8/300H CPU has two operating modes: normal and advanced. Normal mode supports a maximum 64-kbyte address space. Advanced mode supports up to 16 Mbytes. The mode is selected at the mode pins of the microcontroller. For further information, refer to the relevant hardware manual.

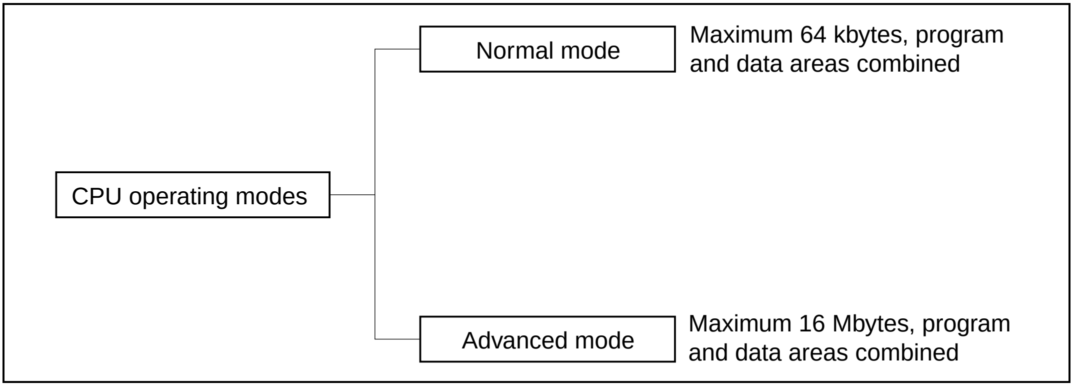


Figure 1.1 CPU Operating Modes

(1) Normal Mode

The exception vector table and stack have the same structure as in the H8/300 CPU.

Address Space: A maximum address space of 64 kbytes can be accessed, as in the H8/300 CPU.

Extended Registers (En): The extended registers (E0 to E7) can be used as 16-bit data registers, or they can be combined with the general registers (R0 to R7) for use as 32-bit data registers. When En is used as a 16-bit register it can contain any value, even when the corresponding general register (R0 to R7) is used as an address register. If the general register is referenced in the register indirect addressing mode with pre-decrement (@–Rn) or post-increment (@Rn+) and a carry or borrow occurs, however, the value in the corresponding extended register will be affected.

Instruction Set: All additional instructions and addressing modes of the H8/300 CPU can be used. If a 24-bit effective address (EA) is specified, only the lower 16 bits are used.

Exception Vector Table and Memory Indirect Branch Addresses: In normal mode the top area starting at H'0000 is allocated to the exception vector table. One branch address is stored per 16 bits (figure 1.2). The exception vector table differs depending on the microcontroller, so see the microcontroller hardware manual for further information.

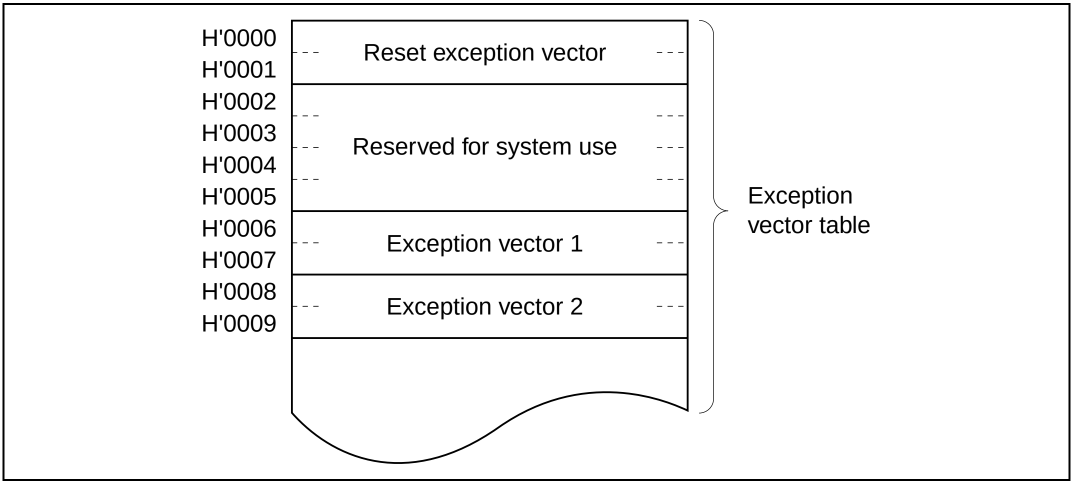


Figure 1.2 Exception Vector Table (normal mode)

The memory indirect addressing mode (@@aa:8) employed in the JMP and JSR instructions uses an 8-bit absolute address to specify a memory operand that contains a branch address. In normal mode the operand is a 16-bit word operand, providing a 16-bit branch address. Branch addresses can be stored in the top area from H'0000 to H'00FF. Note that this area is also used for the exception vector table.

Stack Structure: When the program counter (PC) is pushed on the stack in a subroutine call, and the PC and condition-code register (CCR) are pushed on the stack in exception handling, they are stored in the same way as in the H8/300 CPU. See figure 1.3.

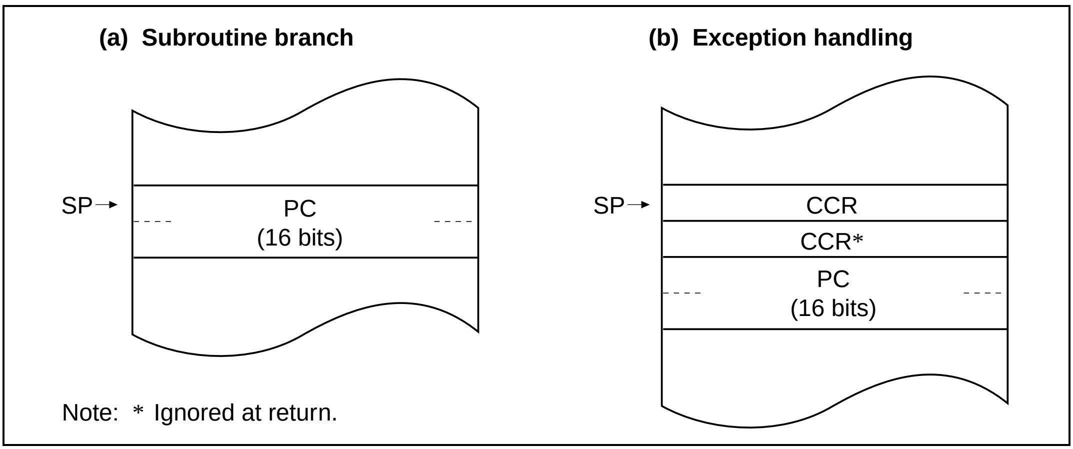


Figure 1.3 Stack Structure (normal mode)

(2) Advanced Mode

In advanced mode the exception vector table and stack structure differ from the H8/300 CPU.

Address Space: Up to 16 Mbytes can be accessed linearly.

Extended Registers (En): The extended registers (E0 to E7) can be used as 16-bit data registers, or they can be combined with the general registers (R0 to R7) for use as 32-bit data registers. When a 32-bit register is used as an address register, the upper 8 bits are ignored.

Instruction Set: All additional instructions and addressing modes of the H8/300H can be used.

Exception Vector Table and Memory Indirect Branch Addresses: In advanced mode the top area starting at H'000000 is allocated to the exception vector table in units of 32 bits. In each 32 bits, the upper 8 bits are ignored and a branch address is stored in the lower 24 bits (figure 1.4). The exception vector table differs depending on the microcontroller, so see the relevant hardware manual for further information.

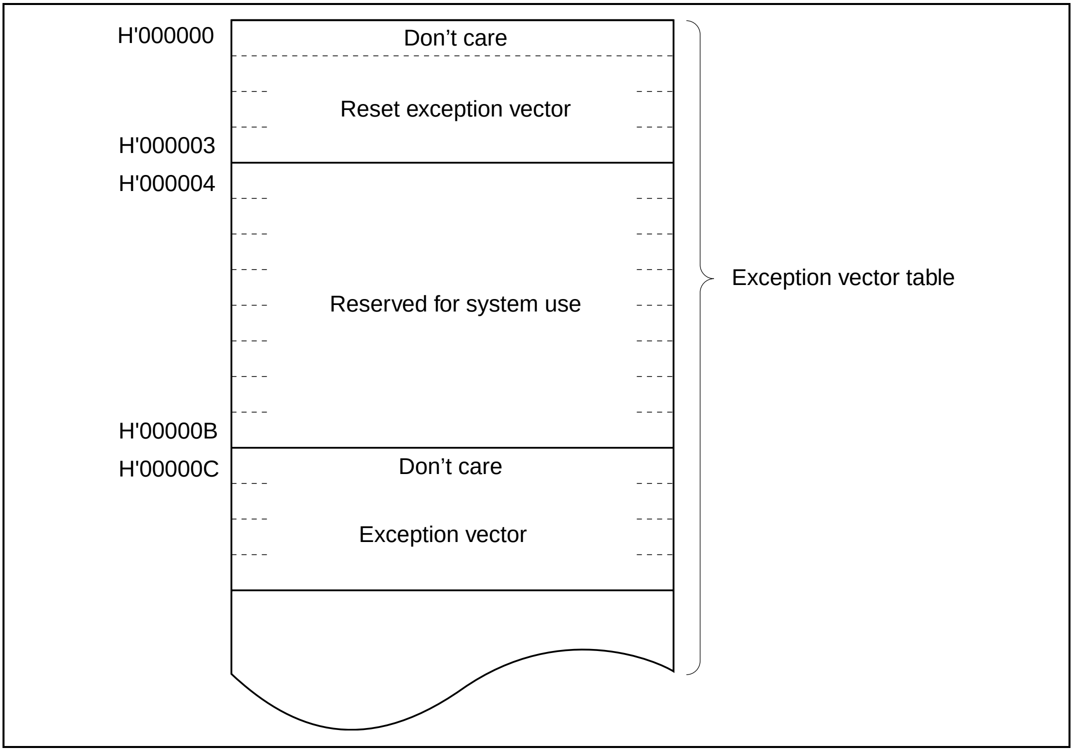


Figure 1.4 Exception Vector Table (advanced mode)

The memory indirect addressing mode (@@aa:8) employed in the JMP and JSR instructions uses an 8-bit absolute address to specify a memory operand that contains a branch address. In advanced mode the operand is a 32-bit longword operand, of which the lower 24 bits are the branch address. Branch addresses can be stored in the top area from H'000000 to H'0000FF. Note that this area is also used for the exception vector table.

Stack Structure: When the program counter (PC) is pushed on the stack in a subroutine call, and the PC and condition-code register (CCR) are pushed on the stack in exception handling, they are stored as shown in figure 1.5.

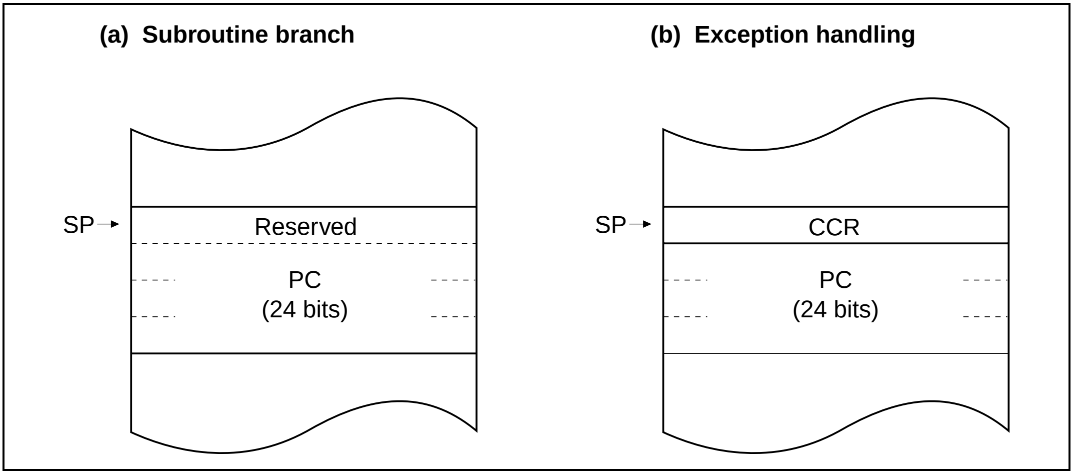


Figure 1.5 Stack Structure (advanced mode)

1.3 Address Space

Figure 1.6 shows a memory map of the H8/300H CPU.

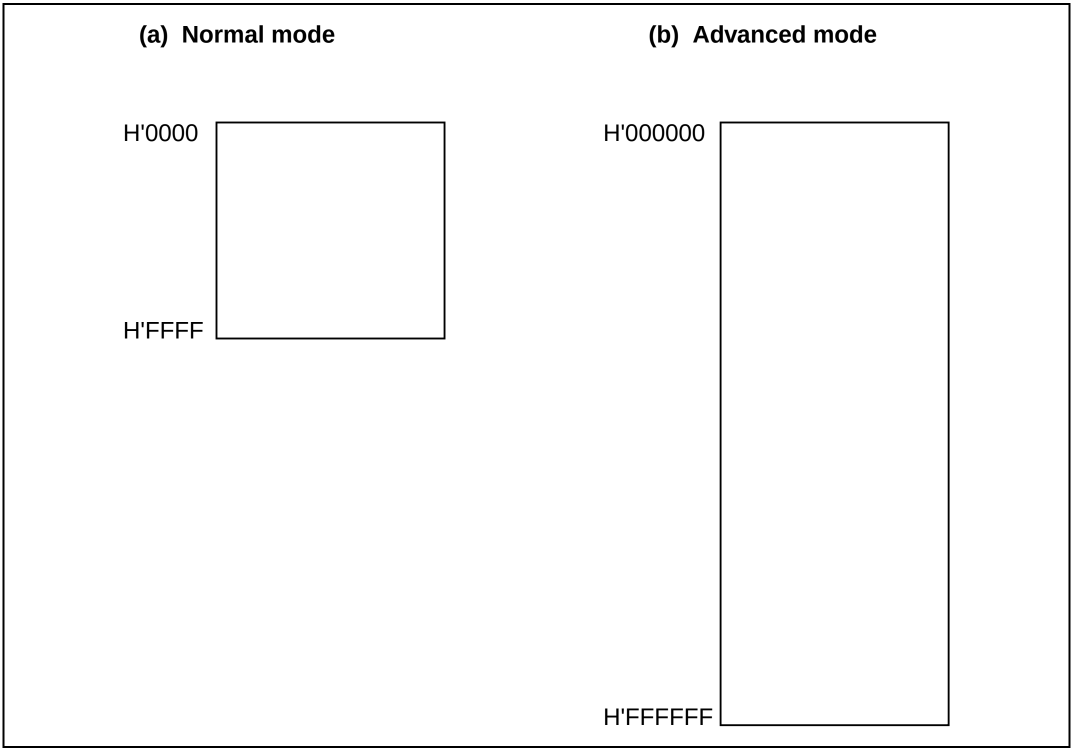


Figure 1.6 Memory Map

1.4 Register Configuration

1.4.1 Overview

The H8/300H CPU has the internal registers shown in figure 1.7. There are two types of registers: general and extended registers, and control registers.

General registers (Rn) and extended registers (En)

	15 … 0	7 … 0	7 … 0
	E0	R0H	R0L
	E1	R1H	R1L
	E2	R2H	R2L
	E3	R3H	R3L
	E4	R4H	R4L
	E5	R5H	R5L
	E6	R6H	R6L
SP	E7	R7H	R7L

Control registers (CR)

23 … 0
PC

	7	6	5	4	3	2	1	0
CCR	I	U	H	U	N	Z	V	C

Legend:

SP: Stack pointer
PC: Program counter
CCR: Condition code register
I: Interrupt mask bit
U: User bit or interrupt mask bit
H: Half-carry flag
N: Negative flag
Z: Zero flag
V: Overflow flag
C: Carry flag

Figure 1.7 CPU Registers

1.4.2 General Registers

The H8/300H CPU has eight 32-bit general registers. These general registers are all functionally alike and can be used without distinction between data registers and address registers. When a general register is used as a data register, it can be accessed as a 32-bit, 16-bit, or 8-bit register. When the general registers are used as 32-bit registers or as address registers, they are designated by the letters ER (ER0 to ER7).

The ER registers divide into 16-bit general registers designated by the letters E (E0 to E7) and R (R0 to R7). These registers are functionally equivalent, providing a maximum sixteen 16-bit registers. The E registers (E0 to E7) are also referred to as extended registers.

The R registers divide into 8-bit general registers designated by the letters RH (R0H to R7H) and RL (R0L to R7L). These registers are functionally equivalent, providing a maximum sixteen 8-bit registers.

Figure 1.8 illustrates the usage of the general registers. The usage of each register can be selected independently.

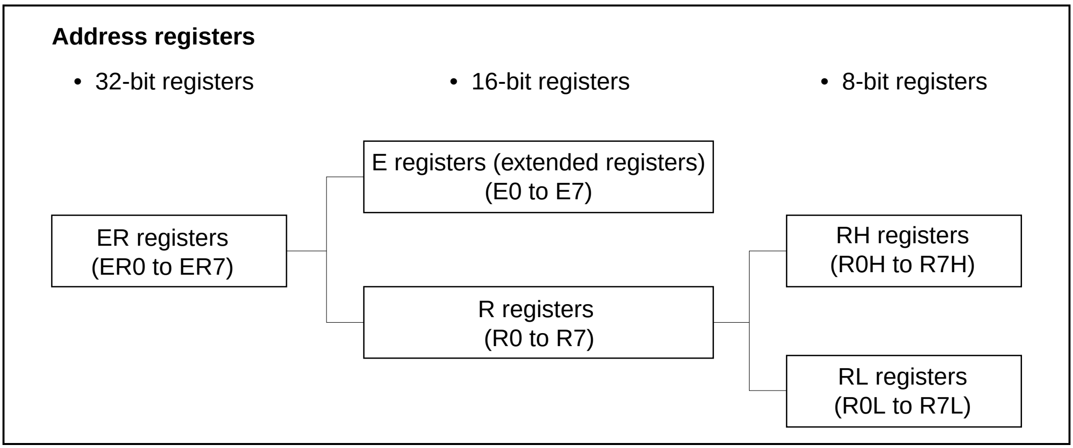


Figure 1.8 Usage of General Registers

General register ER7 has the function of stack pointer (SP) in addition to its general-register function, and is used implicitly in exception handling and subroutine calls. Figure 1.9 shows the stack.

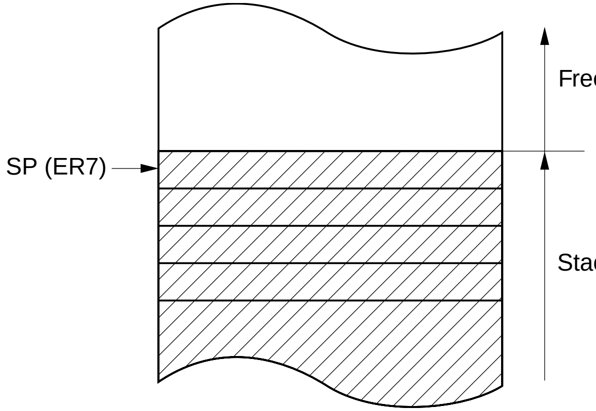


Figure 1.9 Stack

1.4.3 Control Registers

The control registers are the 24-bit program counter (PC) and the 8-bit condition-code register (CCR).

(1) Program Counter (PC)

This 24-bit counter indicates the address of the next instruction the CPU will execute. The length of all CPU instructions is 16 bits (one word) or a multiple of 16 bits, so the least significant PC bit is ignored. When an instruction is fetched, the least significant PC bit is regarded as 0.

(2) Condition Code Register (CCR)

This 8-bit register contains internal CPU status information, including the interrupt mask bit (I) and half-carry (H), negative (N), zero (Z), overflow (V), and carry (C) flags.

Bit 7—Interrupt Mask Bit (I): Masks interrupts other than NMI when set to 1. (NMI is accepted regardless of the I bit setting.) The I bit is set to 1 by hardware at the start of an exception-handling sequence.

Bit 6—User Bit (U): Can be written and read by software using the LDC, STC, ANDC, ORC, and XORC instructions. This bit can also be used as an interrupt mask bit. For details see the relevant microcontroller hardware manual.

Bit 5—Half-Carry Flag (H): When the ADD.B, ADDX.B, SUB.B, SUBX.B, CMP.B, or NEG.B instruction is executed, this flag is set to 1 if there is a carry or borrow at bit 3, and cleared to 0 otherwise. When the ADD.W, SUB.W, CMP.W, or NEG.W instruction is executed, the H flag is set to 1 if there is a carry or borrow at bit 11, and cleared to 0 otherwise. When the ADD.L, SUB.L, CMP.L, or NEG.L instruction is executed, the H flag is set to 1 if there is a carry or borrow at bit 27, and cleared to 0 otherwise.

Bit 4—User Bit (U): Can be written and read by software using the LDC, STC, ANDC, ORC, and XORC instructions.

Bit 3—Negative Flag (N): Indicates the most significant bit (sign bit) of the result of an instruction.

Bit 2—Zero Flag (Z): Set to 1 to indicate a zero result, and cleared to 0 to indicate a non-zero result.

Bit 1—Overflow Flag (V): Set to 1 when an arithmetic overflow occurs, and cleared to 0 at other times.

Bit 0—Carry Flag (C): Set to 1 when a carry occurs, and cleared to 0 otherwise. Used by:

- Add instructions, to indicate a carry
- Subtract instructions, to indicate a borrow
- Shift and rotate instructions, to store the value shifted out of the end bit

The carry flag is also used as a bit accumulator by bit manipulation instructions. Some instructions leave some or all of the flag bits unchanged. For the action of each instruction on the flag bits, refer to the detailed descriptions of the instructions starting in section 2.2.1.

Operations can be performed on the CCR bits by the LDC, STC, ANDC, ORC, and XORC instructions. The N, Z, V, and C flags are used as branching conditions for conditional branch (Bcc) instructions.

1.4.4 Initial Register Values

When the CPU is reset, the program counter (PC) is loaded from the vector table and the I bit in the condition-code register (CCR) is set to 1. The other CCR bits and the general registers and extended registers are not initialized. In particular, the stack pointer (extended register E7 and general register R7) is not initialized. The stack pointer must therefore be initialized by an MOV.L instruction executed immediately after a reset.

1.5 Data Formats

The H8/300H CPU can process 1-bit, 4-bit, 8-bit (byte), 16-bit (word), and 32-bit (longword) data. Bit-manipulation instructions operate on 1-bit data by accessing bit n (n = 0, 1, 2, …, 7) of byte operand data. The DAA and DAS decimal-adjust instructions treat byte data as two digits of 4-bit BCD data.

1.5.1 General Register Data Formats

Figure 1.10 shows the data formats in general registers.

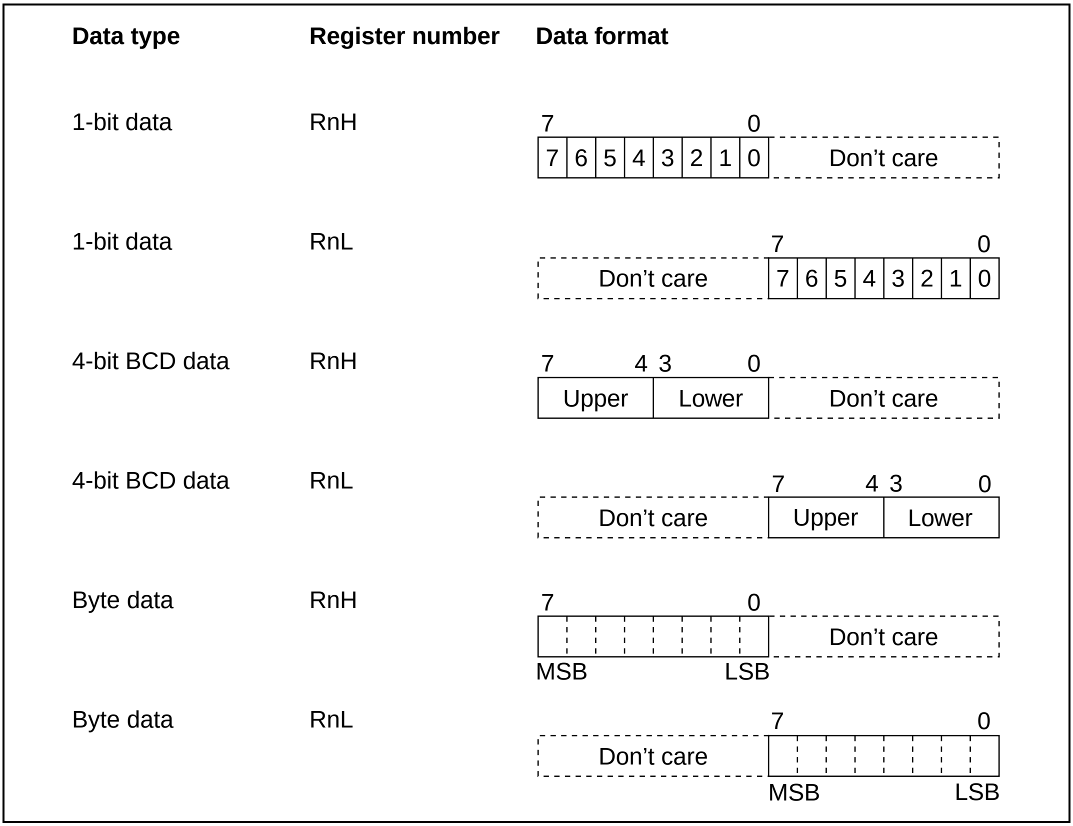
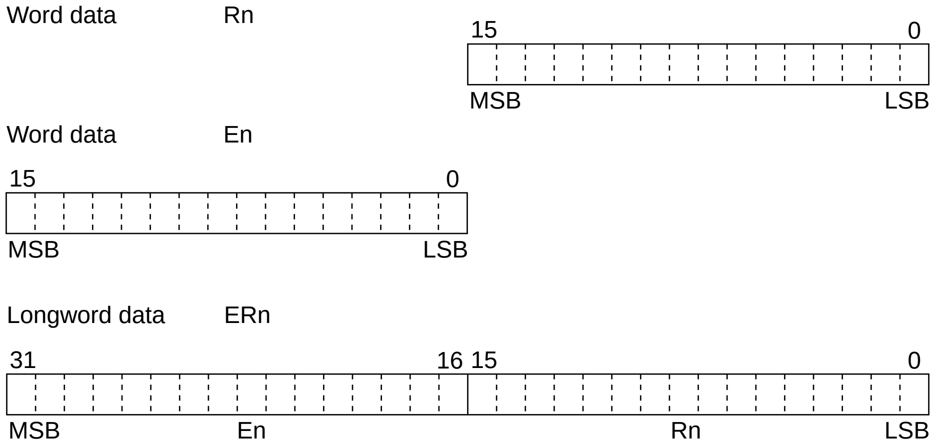


Figure 1.10 General Register Data Formats



Legend:

ERn: General register ER
En: General register E
Rn: General register R
RnH: General register RH
RnL: General register RL
MSB: Most significant bit
LSB: Least significant bit

Figure 1.10 General Register Data Formats (cont)

1.5.2 Memory Data Formats

Figure 1.11 shows the data formats on memory. The H8/300H CPU can access word data and longword data on memory, but word or longword data must begin at an even address. If an attempt is made to access word or longword data at an odd address, no address error occurs but the least significant bit of the address is regarded as 0, so the access starts at the preceding address. This also applies to instruction fetches.

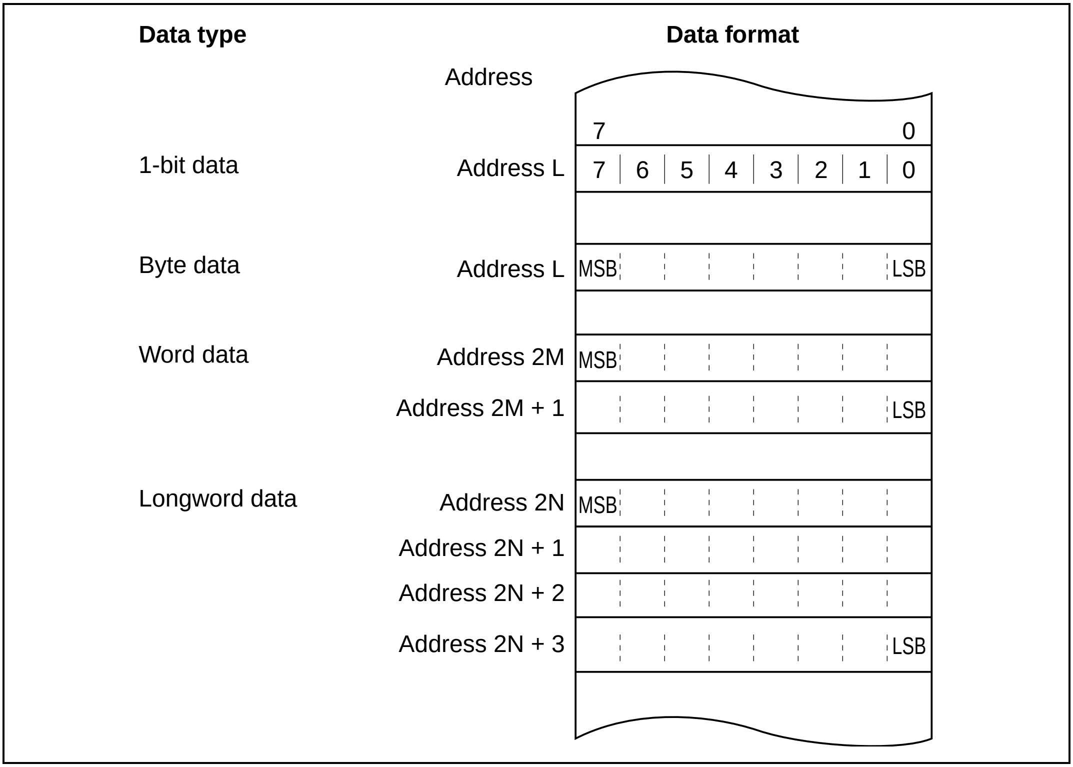


Figure 1.11 Memory Data Formats

When ER7 is used as an address register to access the stack, the operand size should be word size or longword size.

1.6 Instruction Set

1.6.1 Overview

The H8/300H CPU has 62 types of instructions, which are classified by function in table 1.1. For a detailed description of each instruction see section 2.2, Instruction Descriptions.

Table 1.1 Instruction Classification

Function	Instructions	Number
Data transfer	MOV, PUSH*1, POP*2, MOVTPE, MOVFPE	3
Arithmetic operations	ADD, SUB, ADDX, SUBX, INC, DEC, ADDS, SUBS, DAA, DAS, MULXU, MULXS, DIVXU, DIVXS, CMP, NEG, EXTS, EXTU	18
Logic operations	AND, OR, XOR, NOT	4
Shift	SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL, ROTXR	8
Bit manipulation	BSET, BCLR, BNOT, BTST, BAND, BIAND, BOR, BIOR, BXOR, BIXOR, BLD, BILD, BST, BIST	14
Branch	Bcc*2, JMP, BSR, JSR, RTS	5
System control	TRAPA, RTE, SLEEP, LDC, STC, ANDC, ORC, XORC, NOP	9
Block data transfer	EEPMOV	1
	Total 62 types	

Notes: The shaded instructions are not present in the H8/300 instruction set.

1. POP.W Rn and PUSH.W Rn are identical to MOV.W @SP+, Rn and MOV.W Rn, @–SP. POP.L ERn and PUSH.L ERn are identical to MOV.L @SP+, ERn and MOV.L ERn, @–SP.
2. Bcc is the generic designation of a conditional branch instruction.

1.6.2 Instructions and Addressing Modes

Table 1.2 indicates the instructions available in the H8/300H CPU.

Table 1.2 Instruction Set Overview

Function	Instruction	Addressing Modes												
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:24,ERn)	@ERn+/@–ERn	@aa:8	@aa:16	@aa:24	@(d:8,PC)	@(d:16,PC)	@@aa:8	—
Data transfer	MOV	BWL	BWL	BWL	BWL	BWL	BWL	B	BWL	BWL	—	—	—	—
	POP, PUSH	—	—	—	—	—	—	—	—	—	—	—	—	WL
	MOVFPE, MOVTPE	—	—	—	—	—	—	—	B	—	—	—	—	
Arithmetic operations	ADD, CMP	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—
	SUB	WL	BWL	—	—	—	—	—	—	—	—	—	—	—
	ADDX, SUBX	B	B	—	—	—	—	—	—	—	—	—	—	—
	ADDS, SUBS	—	L*[1]	—	—	—	—	—	—	—	—	—	—	—
	INC, DEC	—	BWL	—	—	—	—	—	—	—	—	—	—	—
	DAA, DAS	—	B	—	—	—	—	—	—	—	—	—	—	—
	MULXU, DIVXU	—	BW	—	—	—	—	—	—	—	—	—	—	—
	MULXS, DIVXS	—	BW	—	—	—	—	—	—	—	—	—	—	—
	NEG	—	BWL	—	—	—	—	—	—	—	—	—	—	—
	EXTU, EXTS	—	WL	—	—	—	—	—	—	—	—	—	—	—
Logic operations	AND, OR, XOR	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—
	NOT	—	BWL	—	—	—	—	—	—	—	—	—	—	—
Shift		—	BWL	—	—	—	—	—	—	—	—	—	—	—
Bit manipulation		—	B	B	—	—	—	B	—	—	—	—	—	—

Function	Instruction	Addressing Modes												
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:24,ERn)	@ERn+/@–ERn	@aa:8	@aa:16	@aa:24	@(d:8,PC)	@(d:16,PC)	@@aa:8	—
Branch	Bcc, BSR	—	—	—	—	—	—	—	—	—	○	○	—	—
	JMP, JSR	—	—	○	—	—	—	—	—	○*2	—	—	○	—
	RTS	—	—	—	—	—	—	—	—	—	—	—	—	○
System control	TRAPA	—	—	—	—	—	—	—	—	—	—	—	—	○
	RTE	—	—	—	—	—	—	—	—	—	—	—	—	○
	SLEEP	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDC	B	B	W	W	W	W	—	W	W	—	—	—	—
	STC	—	B	W	W	W	W	—	W	W	—	—	—	—
	ANDC, ORC, XORC	B	—	—	—	—	—	—	—	—	—	—	—	—
	NOP	—	—	—	—	—	—	—	—	—	—	—	—	○
Block data transfer	EEPMOV.B	—	—	—	—	—	—	—	—	—	—	—	—	○
	EEPMOV.W	—	—	—	—	—	—	—	—	—	—	—	—	○

Legend:

B: Byte

W: Word

L: Longword

(shaded) : Newly added instruction in H8/300H CPU

Notes: 1. The operand size of the ADDS and SUBS instructions of the H8/300H CPU has been changed to longword size. (In the H8/300 CPU it was word size.)

2. Because of its larger address space, the H8/300H CPU uses a 24-bit absolute address for the JMP and JSR instructions. (The H8/300 CPU used 16 bits.)

1.6.3 Tables of Instructions Classified by Function

Table 1.3 summarizes the instructions in each functional category. The notation used in table 1.3 is defined next.

Operation Notation

Rd	General register (destination)*
Rs	General register (source)*
Rn	General register*
ERn	General register (32-bit register)
(EAd)	Destination operand
(EAs)	Source operand
CCR	Condition code register
N	N (negative) bit of CCR
Z	Z (zero) bit of CCR
V	V (overflow) bit of CCR
C	C (carry) bit of CCR
PC	Program counter
SP	Stack pointer
#IMM	Immediate data
disp	Displacement
+	Addition
–	Subtraction
×	Multiplication
÷	Division
∧	AND logical
∨	OR logical
⊕	Exclusive OR logical
→	Move
¬	Not
:3/:8/:16/:24	3-, 8-, 16-, or 24-bit length

Note: * General registers include 8-bit registers (R0H/R0L to R7H/R7L), 16-bit registers (R0 to R7, E0 to E7), and 32-bit registers (ER0 to ER7).

Table 1.3 Instructions Classified by Function

Type	Instruction	Size*	Function
Data transfer	MOV	B/W/L	(EAs) → Rd, Rs → (EAd) Moves data between two general registers or between a general register and memory, or moves immediate data to a general register.
	MOVFPE	B	(EAs) → Rd Moves external memory contents (addressed by @aa:16) to a general register in synchronization with an E clock.
	MOVTPE	B	Rs → (EAd) Moves general register contents to an external memory location (addressed by @aa:16) in synchronization with an E clock.
	POP	W/L	@SP+ → Rn Pops a register from the stack. POP.W Rn is identical to MOV.W @SP+, Rn. POP.L ERn is identical to MOV.L @SP+, ERn.
	PUSH	W/L	Rn → @–SP Pushes a register onto the stack. PUSH.W Rn is identical to MOV.W Rn, @–SP. PUSH.L ERn is identical to MOV.L ERn, @–SP.
Arithmetic operations	ADD SUB	B/W/L	Rd ± Rs → Rd, Rd ± #IMM → Rd Performs addition or subtraction on data in two general registers, or on immediate data and data in a general register. (Immediate byte data cannot be subtracted from data in a general register. Use the SUBX or ADD instruction.)
	ADDX SUBX	B	Rd ± Rs ± C → Rd, Rd ± #IMM ± C → Rd Performs addition or subtraction with carry or borrow on byte data in two general registers, or on immediate data and data in a general register.
	INC DEC	B/W/L	Rd ± 1 → Rd, Rd ± 2 → Rd Increments or decrements a general register by 1 or 2. (Byte operands can be incremented or decremented by 1 only.)

Type	Instruction	Size*	Function
Arithmetic operations	ADDS SUBS	L	Rd ± 1 → Rd, Rd ± 2 → Rd, Rd ± 4 → Rd Adds or subtracts the value 1, 2, or 4 to or from data in a 32-bit register.
	DAA DAS	B	Rd decimal adjust → Rd Decimal-adjusts an addition or subtraction result in a general register by referring to the CCR to produce 4-bit BCD data.
	MULXS	B/W	Rd × Rs → Rd Performs signed multiplication on data in two general registers: either 8 bits × 8 bits → 16 bits or 16 bits × 16 bits → 32 bits.
	MULXU	B/W	Rd × Rs → Rd Performs unsigned multiplication on data in two general registers: either 8 bits × 8 bits → 16 bits or 16 bits × 16 bits → 32 bits.
	DIVXS	B/W	Rd ÷ Rs → Rd Performs signed division on data in two general registers: either 16 bits ÷ 8 bits → 8-bit quotient and 8-bit remainder or 32 bits ÷ 16 bits → 16-bit quotient and 16-bit remainder.
	DIVXU	B/W	Rd ÷ Rs → Rd Performs unsigned division on data in two general registers: either 16 bits ÷ 8 bits → 8-bit quotient and 8-bit remainder or 32 bits ÷ 16 bits → 16-bit quotient and 16-bit remainder.
	CMP	B/W/L	Rd – Rs, Rd – #IMM Compares data in a general register with data in another general register or with immediate data, and sets the CCR according to the result.
	NEG	B/W/L	0 – Rd → Rd Takes the two's complement (arithmetic complement) of data in a general register.

Type	Instruction	Size*	Function
Arithmetic operations	EXTS	W/L	Rd (sign extension) → Rd Extends byte data in the lower 8 bits of a 16-bit register to word data, or extends word data in the lower 16 bits of a 32-bit register to longword data, by extending the sign bit.
	EXTU	W/L	Rd (zero extension) → Rd Extends byte data in the lower 8 bits of a 16-bit register to word data, or extends word data in the lower 16 bits of a 32-bit register to longword data, by padding with zeros.
Logic operations	AND	B/W/L	Rd ∧ Rs → Rd, Rd ∧ #IMM → Rd Performs a logical AND operation on a general register and another general register or immediate data.
	OR	B/W/L	Rd ∨ Rs → Rd, Rd ∨ #IMM → Rd Performs a logical OR operation on a general register and another general register or immediate data.
	XOR	B/W/L	Rd ⊕ Rs → Rd, Rd ⊕ #IMM → Rd Performs a logical exclusive OR operation on a general register and another general register or immediate data.
	NOT	B/W/L	¬ (Rd) → (Rd) Takes the one's complement of general register contents.
Shift operations	SHAL SHAR	B/W/L	Rd (shift) → Rd Performs an arithmetic shift on general register contents.
	SHLL SHLR	B/W/L	Rd (shift) → Rd Performs a logical shift on general register contents.
	ROTL ROTR	B/W/L	Rd (rotate) → Rd Rotates general register contents.
	ROTXL ROTXR	B/W/L	Rd (rotate) → Rd Rotates general register contents through the carry bit.

Type	Instruction	Size*	Function
Bit-manipulation instructions	BSET	B	1 → (<bit-No.> of <EAd>) Sets a specified bit in a general register or memory operand to 1. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BCLR	B	0 → (<bit-No.> of <EAd>) Clears a specified bit in a general register or memory operand to 0. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BNOT	B	¬ (<bit-No.> of <EAd>) → (<bit-No.> of <EAd>) Inverts a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BTST	B	¬ (<bit-No.> of <EAd>) → Z Tests a specified bit in a general register or memory operand and sets or clears the Z flag accordingly. The bit number is specified by 3-bit immediate data or the lower three bits of a general register.
	BAND	B	C ∧ (<bit-No.> of <EAd>) → C ANDs the carry flag with a specified bit in a general register or memory operand and stores the result in the carry flag.
	BIAND	B	C ∧ ¬ (<bit-No.> of <EAd>) → C ANDs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag. The bit number is specified by 3-bit immediate data.

Type	Instruction	Size*	Function
Bit-manipulation instructions	BOR	B	C ∨ (<bit-No.> of <EAd>) → C ORs the carry flag with a specified bit in a general register or memory operand and stores the result in the carry flag.
	BIOR	B	C ∨ [¬ (<bit-No.> of <EAd>)] → C ORs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag. The bit number is specified by 3-bit immediate data.
	BXOR	B	C ⊕ (<bit-No.> of <EAd>) → C Exclusive-ORs the carry flag with a specified bit in a general register or memory operand and stores the result in the carry flag.
	BIXOR	B	C ⊕ [¬ (<bit-No.> of <EAd>)] → C Exclusive-ORs the carry flag with the inverse of a specified bit in a general register or memory operand and stores the result in the carry flag. The bit number is specified by 3-bit immediate data.
	BLD	B	(<bit-No.> of <EAd>) → C Transfers a specified bit in a general register or memory operand to the carry flag.
	BILD	B	¬ (<bit-No.> of <EAd>) → C Transfers the inverse of a specified bit in a general register or memory operand to the carry flag. The bit number is specified by 3-bit immediate data.
	BST	B	C → (<bit-No.> of <EAd>) Transfers the carry flag value to a specified bit in a general register or memory operand.
	BIST	B	¬ C → (<bit-No.> of <EAd>) Transfers the inverse of the carry flag value to a specified bit in a general register or memory operand. The bit number is specified by 3-bit immediate data.

Type	Instruction	Size*	Function
Branching instructions	Bcc	—	Branches to a specified address if a specified condition is true. The branching conditions are listed below.

Mnemonic	Description	Condition
BRA(BT)	Always (true)	Always
BRN(BF)	Never (false)	Never
BHI	High	$C \vee Z = 0$
BLS	Low or same	$C \vee Z = 1$
Bcc(BHS)	Carry clear (high or same)	$C = 0$
BCS(BLO)	Carry set (low)	$C = 1$
BNE	Not equal	$Z = 0$
BEQ	Equal	$Z = 1$
BVC	Overflow clear	$V = 0$
BVS	Overflow set	$V = 1$
BPL	Plus	$N = 0$
BMI	Minus	$N = 1$
BGE	Greater or equal	$N \oplus V = 0$
BLT	Less than	$N \oplus V = 1$
BGT	Greater than	$Z \vee (N \oplus V) = 0$
BLE	Less or equal	$Z \vee (N \oplus V) = 1$

Type	Instruction	Size*	Function
Branching instructions	JMP	—	Branches unconditionally to a specified address.
	BSR	—	Branches to a subroutine at a specified address.
	JSR	—	Branches to a subroutine at a specified address.
	RTS	—	Returns from a subroutine.

Type	Instruction	Size*	Function
System control instructions	TRAPA	—	Starts trap-instruction exception handling.
	RTE	—	Returns from an exception-handling routine.
	SLEEP	—	Causes a transition to the power-down state.
	LDC	B/W	(EAs) → CCR Moves the source operand contents to the condition code register. Byte transfer is performed in the #xx:8, Rs addressing mode and word transfer in other addressing modes.
	STC	B/W	CCR → (EAd) Transfers the CCR contents to a destination location. Byte transfer is performed in the Rd addressing mode and word transfer in other addressing modes.
	ANDC	B	CCR ∧ #IMM → CCR Logically ANDs the condition code register with immediate data.
	ORC	B	CCR ∨ #IMM → CCR Logically ORs the condition code register with immediate data.
	XORC	B	CCR ⊕ #IMM → CCR Logically exclusive-ORs the condition code register with immediate data.
	NOP	—	PC + 2 → PC Only increments the program counter.

Type	Instruction	Size*	Function
Block data transfer instruction	EEPMOV.B	—	if R4L ≠ 0 then Repeat @ER5 +→ @ER6 + R4L – 1→R4L Until R4L = 0 else next;
	EEPMOV.W	—	if R4 ≠ 0 then Repeat @ER5 +→ @ER6 + R4 – 1→R4L Until R4 = 0 else next; Transfers a data block according to parameters set in general registers R4L or R4, ER5, and R6. R4L or R4: size of block (bytes) ER5: starting source address R6: starting destination address Execution of the next instruction begins as soon as the transfer is completed.

Note: * Size refers to the operand size.
B: Byte
W: Word
L: Longword

1.6.4 Basic Instruction Formats

The H8/300H instructions consist of 2-byte (1-word) units. An instruction consists of an operation field (OP field), a register field (r field), an effective address extension (EA field), and a condition field (cc).

Operation Field: Indicates the function of the instruction, the effective address, and the operation to be carried out on the operand. The operation field always includes the first four bits of the instruction. Some instructions have two operation fields.

Register Field: Specifies a general register. Address registers are specified by 3 bits, data registers by 3 bits or 4 bits. Some instructions have two register fields. Some have no register field.

Effective Address Extension: Eight, 16, or 32 bits specifying immediate data, an absolute address, or a displacement. A 24-bit address or a displacement is treated as 32-bit data in which the first 8 bits are 0.

Condition Field: Specifies the branching condition of Bcc instructions.

Figure 1.12 shows examples of instruction formats.

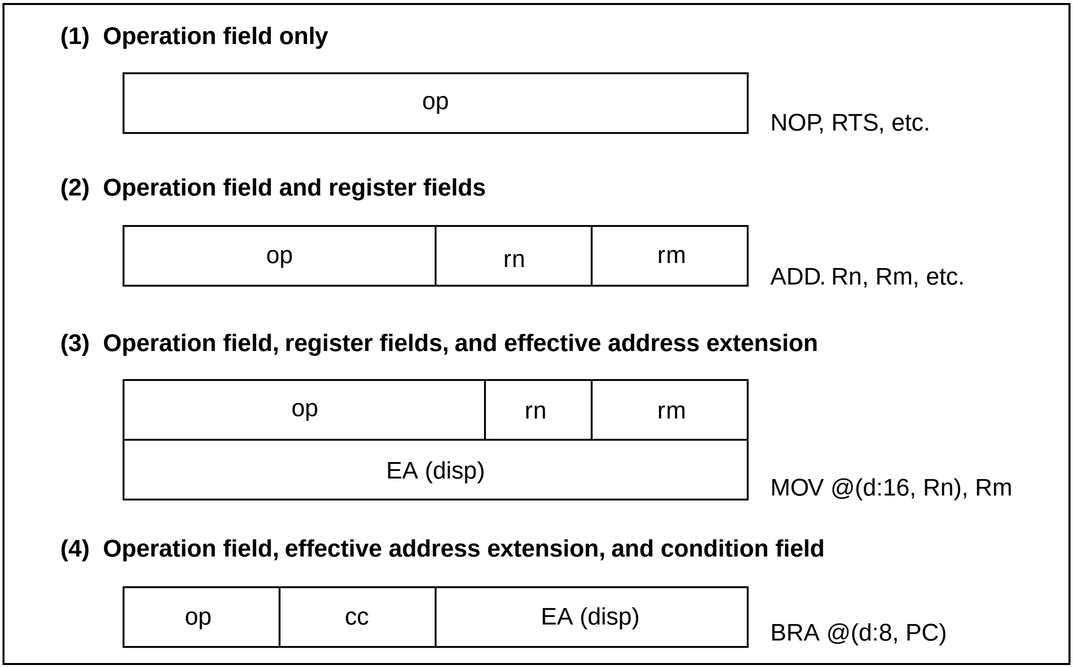


Figure 1.12 Instruction Formats

1.6.5 Addressing Modes and Effective Address Calculation

(1) Addressing Modes

The H8/300H CPU supports the eight addressing modes listed in table 1.4. Each instruction uses a subset of these addressing modes. Arithmetic and logic instructions can use the register direct and immediate modes. Data transfer instructions can use all addressing modes except program-counter relative and memory indirect. Bit manipulation instructions use register direct, register indirect, or absolute (8-bit) addressing mode to specify an operand, and register direct (BSET, BCLR, BNOT, and BTST instructions) or immediate (3-bit) addressing mode to specify a bit number in the operand.

Table 1.4 Addressing Modes

No.	Addressing Mode	Symbol
1	Register direct	Rn
2	Register indirect	@ERn
3	Register indirect with displacement	@(d:16,ERn)/@(d:24,ERn)
4	Register indirect with post-increment Register indirect with pre-decrement	@ERn+ @–ERn
5	Absolute address	@aa:8/@aa:16/@aa:24
6	Immediate	#xx:8/#xx:16/#xx:32
7	Program-counter relative	@(d:8,PC)/@(d:16,PC)
8	Memory indirect	@@aa:8

1 Register Direct—Rn: The register field of the instruction specifies an 8-, 16-, or 32-bit general register containing the operand. R0H to R7H and R0L to R7L can be specified as 8-bit registers. R0 to R7 and E0 to E7 can be specified as 16-bit registers. ER0 to ER7 can be specified as 32-bit registers.

2 Register Indirect—@ERn: The register field of the instruction code specifies an address register (ERn), the lower 24 bits of which contain the address of a memory operand.

3 Register Indirect with Displacement—@(d:16, ERn) or @(d:24, ERn): A 16-bit or 24-bit displacement contained in the instruction is added to an address register (an extended register paired with a general register) specified by the register field of the instruction, and the lower 24 bits of the sum specify the address of a memory operand. A 16-bit displacement is sign-extended when added.

4 Register Indirect with Post-Increment or Pre-Decrement—@ERn+ or @–ERn:

- Register indirect with post-increment—@ERn+

  The register field of the instruction code specifies an address register (ERn), the lower 24 bits of which contain the address of a memory operand. After the operand is accessed, 1, 2, or 4 is added to the address register contents (32 bits) and the sum is stored in the address register. The value added is 1 for byte access, 2 for word access, or 4 for longword access. For word or longword access, the register value should be even.

- Register indirect with pre-decrement—@–ERn

  The value 1, 2, or 4 is subtracted from an address register (ERn) specified by the register field in the instruction code, and the lower 24 bits of the result becomes the address of a memory operand. The result is also stored in the address register. The value subtracted is 1 for byte access, 2 for word access, or 4 for longword access. For word or longword access, the resulting register value should be even.

5 Absolute Address—@aa:8, @aa:16, or @aa:24: The instruction code contains the absolute address of a memory operand. The absolute address may be 8 bits long (@aa:8), 16 bits long (@aa:16), or 24 bits long (@aa:24). For an 8-bit absolute address, the upper 16 bits are all assumed to be 1 (H'FFFF). For a 16-bit absolute address the upper 8 bits are a sign extension. A 24-bit absolute address can access the entire address space. Table 1.5 indicates the accessible address ranges.

Table 1.5 Absolute Address Access Ranges

	Normal Mode	Advanced Mode
8 bits (@aa:8)	H'FF00 to H'FFFF (65,280 to 65,535)	H'FFFF00 to H'FFFFF (16,776,960 to 16,777,215)
16 bits (@aa:16)	H'0000 to H'FFFF (0 to 65,535)	H'000000 to H'007FFF, H'FF8000 to H'FFFFFF (0 to 32,767, 16,744,448 to 16,777,215)
24 bits (@aa:24)	H'0000 to H'FFFF (0 to 65,535)	H'00000 to H'FFFFF (0 to 16,777,215)

For further details on the accessible range, see the relevant microcontroller hardware manual.

6 Immediate—#xx:8, #xx:16, or #xx:32: The instruction contains 8-bit (#xx:8), 16-bit (#xx:16), or 32-bit (#xx:32) immediate data as an operand.

The ADDS, SUBS, INC, and DEC instructions contain immediate data implicitly. Some bit manipulation instructions contain 3-bit immediate data in the second or fourth byte of the instruction, specifying a bit number. The TRAPA instruction contains 2-bit immediate data in the second byte of the instruction, specifying a vector address.

7 Program-Counter Relative—@(d:8, PC) or @(d:16, PC): This mode is used in the Bcc and BSR instructions. An 8-bit or 16-bit displacement contained in the instruction is sign-extended and added to the 24-bit program counter (PC) contents to generate a branch address. The PC value to which the displacement is added is the address of the first byte of the next instruction, so the possible branching range is –126 to +128 bytes (–63 to +64 words) or –32766 to +32768 bytes (–16383 to +16384 words) from the branch instruction. The resulting value should be an even number.

8 Memory Indirect—@@aa:8: This mode can be used by the JMP and JSR instructions. The second byte of the instruction specifies a memory operand by an 8-bit absolute address. This memory operand contains a branch address. The upper 8 bits of the absolute address are assumed to be 0 (H'00), so the address range is 0 to 255 (H’0000 to H’00FF in normal mode, H'000000 to H'0000FF in advanced mode). In normal mode the memory operand is a word operand and the branch address is 16 bits long. In advanced mode the memory operand is a longword operand. The first byte is ignored and the branch address is 24 bits long. Note that the first part of the address range is also the exception vector area. For further details see the relevant microcontroller hardware manual.

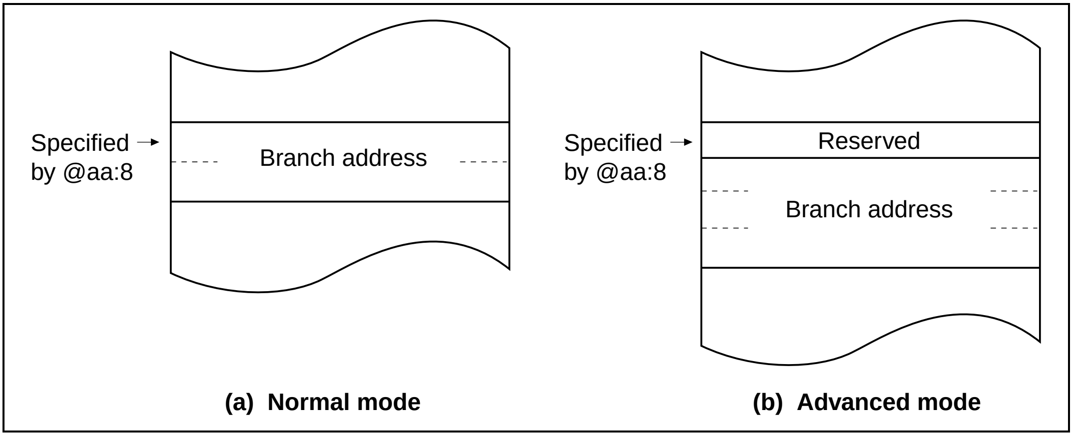


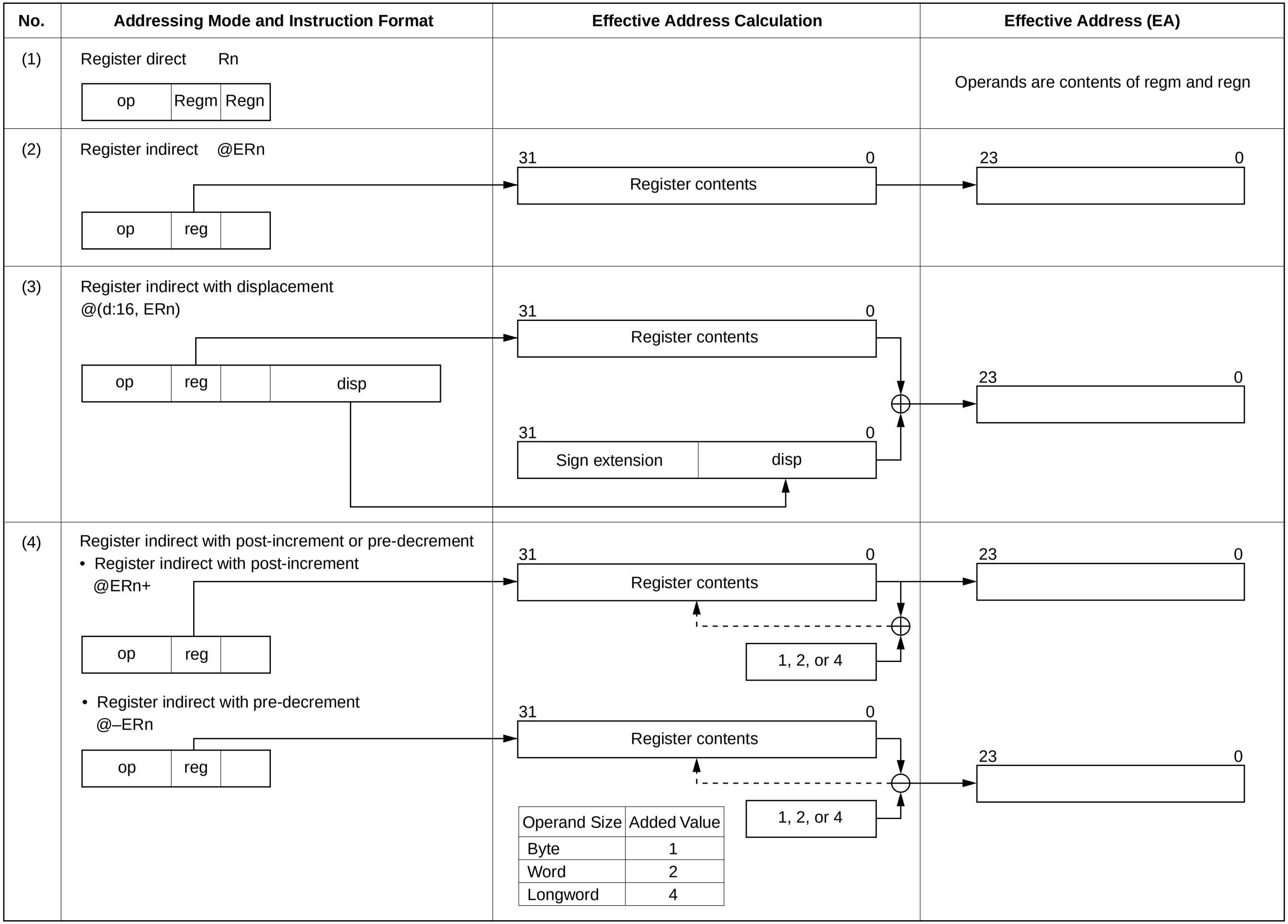
Figure 1.13 Branch Address Specification in Memory Indirect Mode

If an odd address is specified in word or longword memory access, or as a branch address, the least significant bit is regarded as 0, causing access to be performed at the address preceding the specified address. [See (2) Memory Data Formats in section 1.5.2 for further information.]

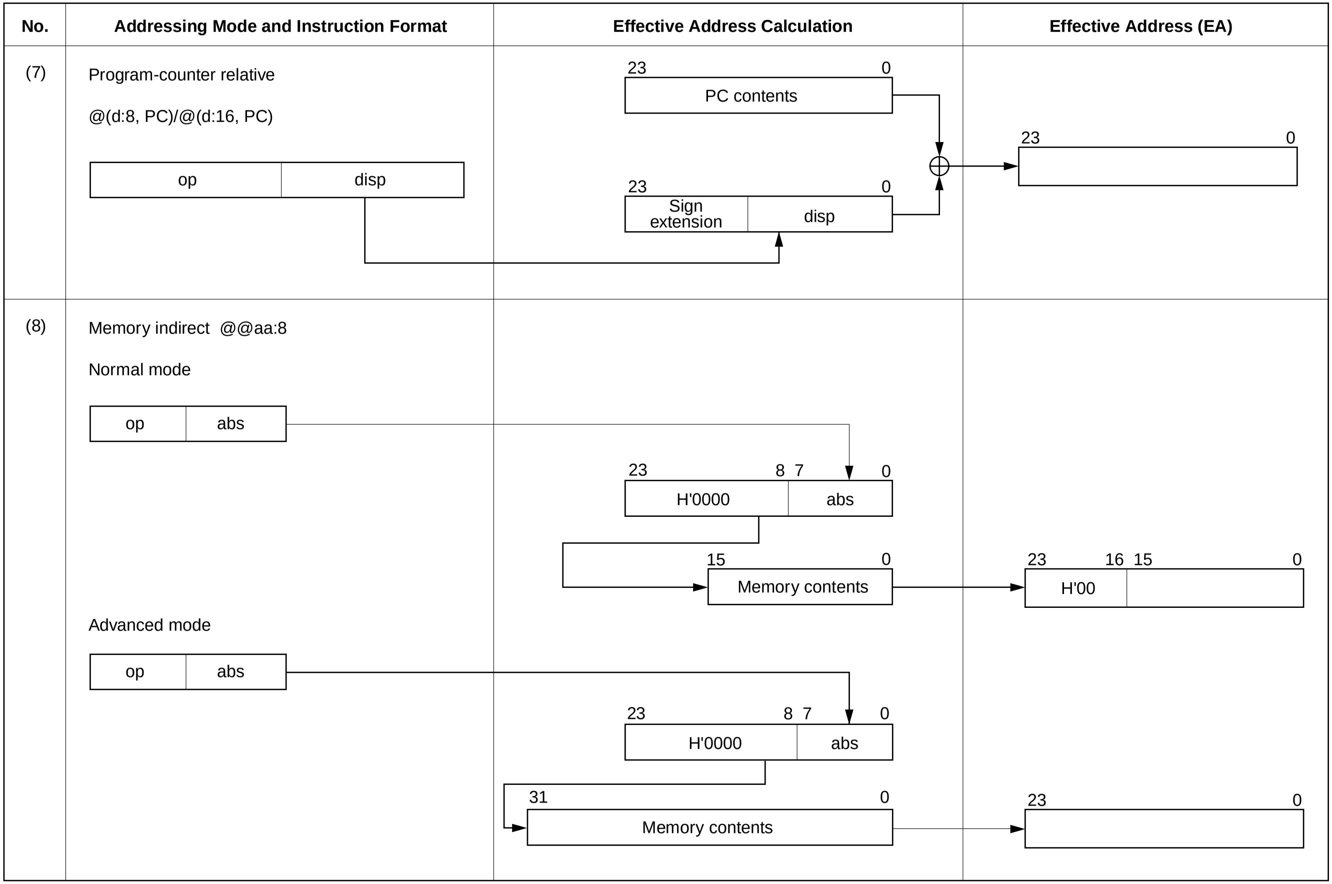
(2) Effective Address Calculation

Table 1.6 indicates how effective addresses are calculated in each addressing mode. In normal mode the upper 8 bits of the effective address are ignored in order to generate a 16-bit address.

Table 1.6 Effective Address Calculation



No.	Addressing Mode and Instruction Format	Effective Address Calculation	Effective Address (EA)
(5)	Absolute address @aa:8 op | abs @aa:16 op | abs @aa:24 op | abs		23 … 8 7 … 0 H'FFFF | 23 … 16 15 … 0 Sign extension | 23 … 0
(6)	Immediate #xx:8/#xx:16/#xx:32 op | IMM		Operand is immediate data.



Legend:
reg, regm, regn: General registers
op: Operation field
disp: Displacement
abs: Absolute address
IMM: Immediate data

Section 2 Instruction Descriptions

2.1 Tables and Symbols

This section explains how to read the tables describing each instruction. Note that the descriptions of some instructions extend over two pages or more.

Mnemonic (full name): Gives the full and mnemonic names of the instruction.

Type: Indicates the type of instruction.

Operation: Describes the instruction in symbolic notation. (See section 2.1.2, Operation.)

Assembly-Language Format: Indicates the assembly-language format of the instruction. (See section 2.1.1, Assembler Format.)

Operand Size: Indicates the available operand sizes.

Condition Code: Indicates the effect of instruction execution on the flag bits in the CCR. (See section 2.1.3, Condition Code.)

Description: Describes the operation of the instruction in detail.

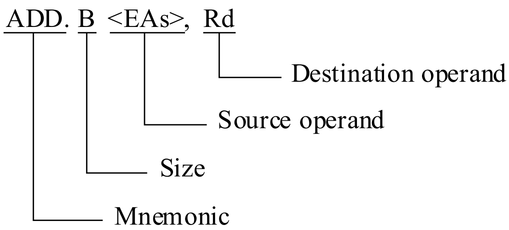
Available Registers: Indicates which registers can be specified in the register field of the instruction.

Operand Format and Number of States Required for Execution: Shows the addressing modes and instruction format together with the number of states required for execution.

Notes: Gives notes concerning execution of the instruction.

2.1.1 Assembler Format

Example:



The operand size is byte (B), word (W), or longword (L). Some instructions are restricted to a limited set of operand sizes.

The symbol <EA> indicates that two or more addressing modes can be used. The H8/300H CPU supports the eight addressing modes listed next. Effective address calculation is described in section 1.7, Effective Address Calculation.

Symbol	Addressing Mode
Rn	Register direct
@ERn	Register indirect
@(d:16, ERn)/@(d:24, ERn)	Register indirect with displacement (16-bit or 24-bit)
@ERn+, @–ERn	Register indirect with post-increment or pre-decrement
@aa:8/16/24	Absolute address (8-bit, 16-bit, or 24-bit)
#xx:8/16/32	Immediate (8-bit, 16-bit, or 32-bit)
@(d:8, PC)/@(d:16, PC)	Program-counter relative (8-bit or 16-bit)
@@aa:8	Memory indirect

2.1.2 Operation

The symbols used in the operation descriptions are defined as follows.

Symbol	Meaning
Rd	General destination register*
Rs	General source register*
Rn	General register*
ERd	General destination register (address register or 32-bit register)
ERs	General source register (address register or 32-bit register)
ERn	General register (32-bit register)
(EAd)	Destination operand
(EAs)	Source operand
PC	Program counter
SP	Stack pointer
CCR	Condition-code register
N	N (negative) flag in CCR
Z	Z (zero) flag in CCR
V	V (overflow) flag in CCR
C	C (carry) flag in CCR
disp	Displacement
→	Transfer from the operand on the left to the operand on the right, or transition from the state on the left to the state on the right
+	Addition of the operands on both sides
–	Subtraction of the operand on the right from the operand on the left
×	Multiplication of the operands on both sides
÷	Division of the operand on the left by the operand on the right
∧	Logical AND of the operands on both sides
∨	Logical OR of the operands on both sides
⊕	Logical exclusive OR of the operands on both sides
¬	Logical NOT (logical complement)
( ) < >	Contents of effective address of the operand

Note: * General registers include 8-bit registers (R0H to R7H and R0L to R7L), 16-bit registers (R0 to R7 ad E0 to E7) and 32-bit registers.

2.1.3 Condition Code

The symbols used in the condition-code description are defined as follows.

Symbol	Meaning
↕	Changes according to the result of the instruction
*	Undetermined (no guaranteed value)
0	Always cleared to 0
—	Not affected by execution of the instruction
Δ	Varies depending on conditions; see the notes.

2.1.4 Instruction Format

The symbols used in the instruction format descriptions are listed below.

Symbol	Meaning
IMM	Immediate data (2, 3, 8, 16, or 32 bits)
abs	Absolute address (8, 16, or 24 bits)
disp	Displacement (8, 16, or 24 bits)
rs, rd, rn	Register number (4 bits. The symbol rs corresponds to operand symbols such as Rs. The symbol rd corresponds to operand symbols such as Rd. The symbol rn corresponds to the operand symbol Rn.)
ers, erd, ern	Register number (3 bits. The symbol ers corresponds to operand symbols such as ERs. The symbol erd corresponds to operand symbols such as ERd and @ERd. The symbol ern corresponds to the operand symbol ERn.)

2.1.5 Register Specification

Address Register Specification: When a general register is used as an address register [@ERn, @(d:16, ERn), @(d:24, ERn), @ERn+, or @–ERn], the register is specified by a 3-bit register field (ers or erd). The lower 24 bits of the register are valid.

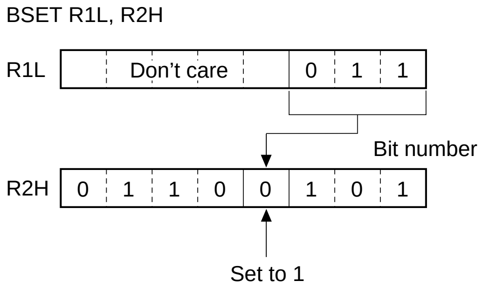
Data Register Specification: A general register can be used as a 32-bit, 16-bit, or 8-bit data register, which is specified by a 3-bit register number. When a 32-bit register (ERn) is used as a longword data register, it is specified by a 3-bit register field (ers, erd, or ern). When a 16-bit register is used as a word data register, it is specified by a 4-bit register field (rs, rd, or rn). The lower 3 bits specify the register number. The upper bit is set to 1 to specify an extended register (En) or cleared to 0 to specify a general register (Rn). When an 8-bit register is used as a byte data register, it is specified by a 4-bit register field (rs, rd, or rn). The lower 3 bits specify the register number. The upper bit is set to 1 to specify a low register (RnL) or cleared to 0 to specify a high register (RnH). This is shown next.

Address Register 32-bit Register		16-bit Register		8-bit Register	
Register Field	General Register	Register Field	General Register	Register Field	General Register
000	ER0	0000	R0	0000	R0H
001	ER1	0001	R1	0001	R1H
⋮	⋮	⋮	⋮	⋮	⋮
111	ER7	0111	R7	0111	R7H
		1000	E0	1000	E0L
		1001	E1	1001	E1L
		⋮	⋮	⋮	⋮
		1111	E7	1111	E7L

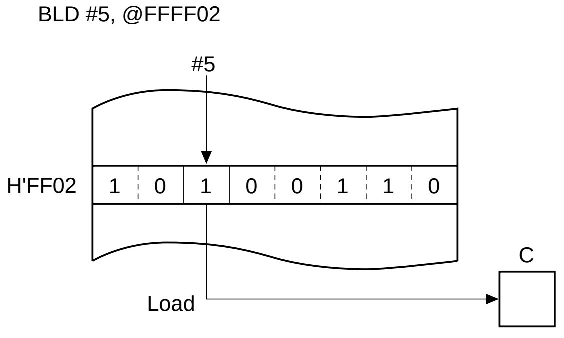
2.1.6 Bit Data Access in Bit Manipulation Instructions

Bit data is accessed as the n-th bit (n = 0, 1, 2, 3, …, 7) of a byte operand in a general register or memory. The bit number is given by 3-bit immediate data, or by the lower 3 bits of a general register value.

Example 1: To set bit 3 in R2H to 1



Example 2: To load bit 5 at address H'FFFF02 into the bit accumulator



The operand size and addressing mode are as indicated for register or memory operand data.

2.2 Instruction Descriptions

The instructions are described starting in section 2.2.1.

2.2.1 (1) ADD (B)

ADD (ADD binary) **Add Binary**

Operation

Rd + (EAs) → Rd

Assembly-Language Format

ADD.B <EAs>, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a carry at bit 3; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a carry at bit 7; otherwise cleared to 0.

Description

This instruction adds the source operand to the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	ADD.B	#xx:8, Rd	8	rd	IMM				2
Register direct	ADD.B	Rs, Rd	0	8	rs	rd			2

Notes

2.2.1 (2) ADD (W)

ADD (ADD binary) **Add Binary**

Operation

Rd + (EAs) → Rd

Assembly-Language Format

ADD.W <EAs>, Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a carry at bit 11; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a carry at bit 15; otherwise cleared to 0.

Description

This instruction adds the source operand to the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format							No. of States
			1st byte		2nd byte		3rd byte		4th byte	
Immediate	ADD.W	#xx:16, Rd	7	9	1	rd	IMM			4
Register direct	ADD.W	Rs, Rd	0	9	rs	rd				2

Notes

2.2.1 (3) ADD (L)

ADD (ADD binary) **Add Binary**

Operation

ERd + (EAs) → ERd

Assembly-Language Format

`ADD.L` <EAs>, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a carry at bit 27; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a carry at bit 31; otherwise cleared to 0.

Description

This instruction adds the source operand to the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format									No. of States
			1st byte		2nd byte			3rd byte	4th byte	5th byte	6th byte	
Immediate	ADD.L	#xx:32, ERd	7	A	1		0 erd	IMM				6
Register direct	ADD.L	Rs, ERd	0	A	1	ers	0 erd					2

Notes

2.2.2 ADDS

ADDS (ADD with Sign extension) **Add Binary Address Data**

Operation

Rd + 1 → ERd
Rd + 2 → ERd
Rd + 4 → ERd

Assembly-Language Format

ADDS #1, ERd
ADDS #2, ERd
ADDS #4, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction adds the immediate value 1, 2, or 4 to the contents of a 32-bit register ERd. Differing from the ADD instruction, it does not affect the condition code flags.

Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format							No. of States
			1st byte		2nd byte			3rd byte	4th byte	
Register direct	ADDS	#1, ERd	0	B	0	0	erd			2
Register direct	ADDS	#2, ERd	0	B	8	0	erd			2
Register direct	ADDS	#4, ERd	0	B	9	0	erd			2

Notes

2.2.3 ADDX

ADDX (ADD with eXtend carry) **Add with Carry**

Operation

Rd + (EAs) + C → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a carry at bit 3; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Previous value remains unchanged if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a carry at bit 7; otherwise cleared to 0.

Assembly-Language Format

ADDX <EAs>, Rd

Operand Size

Byte

Description

This instruction adds the source operand and carry flag to the contents of an 8-bit register Rd (destination register) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	ADDX	#xx:8, Rd	9	rd	IMM				2
Register direct	ADDX	Rs, Rd	0	E	rs	rd			2

Notes

2.2.4 (1) AND (B)

AND (AND logical) **Logical AND**

Operation

Rd ∧ (EAs) → Rd

Assembly-Language Format

`AND.B` <EAs>, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction ANDs the source operand with the contents of an 8-bit register Rd (destination register) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Immediate	AND.B	#xx:8, Rd	E ¦ rd	IMM			2
Register direct	AND.B	Rs, Rd	1 ¦ 6	rs ¦ rd			2

Notes

2.2.4 (2) AND (W)

AND (AND logical) **Logical AND**

Operation

Rd ∧ (EAs) → Rd

Assembly-Language Format

`AND.W` <EAs>, Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction ANDs the source operand with the contents of a 16-bit register Rd (destination register) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	AND.W	#xx:16, Rd	7	9	6	rd	IMM		4
Register direct	AND.W	Rs, Rd	6	6	rs	rd			2

Notes

2.2.4 (3) AND (L)

AND (AND logical) **Logical AND**

Operation

ERd ∧ (EAs) → ERd

Assembly-Language Format

AND.L <EAs>, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction ANDs the source operand with the contents of a 32-bit register ERd (destination register) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	
Immediate	AND.L	#xx:32, ERd	7 A	6 0 erd	IMM				6
Register direct	AND.L	Rs, ERd	0 1	F 0	6 6	0 ers 0 erd			4

Notes

2.2.5 ANDC

ANDC (AND Control register) **Logical AND with CCR**

Operation

CCR ∧ #IMM → CCR

Assembly-Language Format

ANDC #xx:8, CCR

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

I: Stores the corresponding bit of the result.
UI: Stores the corresponding bit of the result
H: Stores the corresponding bit of the result.
U: Stores the corresponding bit of the result
N: Stores the corresponding bit of the result.
Z: Stores the corresponding bit of the result.
V: Stores the corresponding bit of the result.
C: Stores the corresponding bit of the result.

Description

This instruction ANDs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format					No. of States
			1st byte		2nd byte	3rd byte	4th byte	
Immediate	ANDC	#xx:8, CCR	0	6	IMM			2

Notes

2.2.6 BAND

BAND (Bit AND) **Bit Logical AND**

Operation

C ∧ (<bit No.> of <EAd>) → C

Assembly-Language Format

BAND #xx:3, <EAd>

Operand Size

Byte

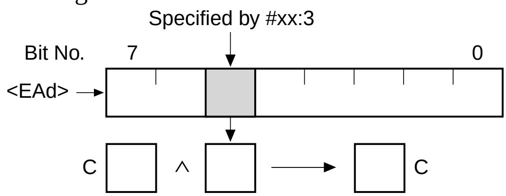
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Stores the result of the operation.

Description

This instruction ANDs a specified bit in the destination operand with the carry bit and stores the result in the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	BAND	#xx:3.Rd	7 6	0 IMM rd			2
Register indirect	BAND	#xx:3.@ERd	7 C	0 erd 0	7 6	0 IMM 0	6
Absolute address	BAND	#xx:3.@aa:8	7 E	abs	7 6	0 IMM 0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

See the corresponding LSI hardware manual for details on the access range for @aa : 8.

2.2.7 Bcc

Bcc (Branch conditionally) **Conditional Branch**

Operation

If condition is true, then

PC + disp → PC

else next;

Assembly-Language Format

Bcc disp

→ Condition field

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

If the condition specified in the condition field (cc) is true, a displacement is added to the program counter (PC) and execution branches to the resulting address. The PC value used in the address calculation is the starting address of the instruction immediately following the Bcc instruction. The displacement is a signed 8-bit or 16-bit value. The branch destination address can be located in the range from –126 to +128 bytes or –32766 to +32768 bytes from the Bcc instruction.

Mnemonic	Meaning	cc	Condition	Signed/Unsigned*
BRA (BT)	Always (true)	0000	True	
BRn (BF)	Never (false)	0001	False	
BHI	HIgh	0010	C ∨ Z = 0	X > Y (unsigned)
BLS	Low or Same	0011	C ∨ Z = 1	X ≤ Y (unsigned)
BCC (BHS)	Carry Clear (High or Same)	0100	C = 0	X ≥ Y (unsigned)
BCS (BLO)	Carry Set (LOw)	0101	C = 1	X < Y (unsigned)
BNE	Not Equal	0110	Z = 0	X ≠ Y (unsigned or signed)
BEQ	EQual	0111	Z = 1	X > Y (unsigned or signed)
BVC	oVerflow Clear	1000	V = 0	
BVS	oVerflow Set	1001	V = 1	
BPL	PLus	1010	N = 0	
BMI	Minus	1011	N = 1	
BGE	Greater or Equal	1100	N ⊕ V = 0	X ≥ Y (signed)
BLT	Less Than	1101	N ⊕ V = 1	X < Y (signed)
BGT	Greater Than	1110	Z ∨ (N ⊕ V) = 0	X > Y (signed)
BLE	Less or Equal	1111	Z ∨ (N ⊕ V) = 1	X ≤ Y (signed)

Note: * If the immediately preceding instruction is a CMP instruction, X is the destination operand and Y is the source operand.

Bcc

Bcc (Branch conditionally) **Conditional Branch**

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Program-counter relative	BRA (BT)	d:8	4	0	disp				4
		d:16	5	8	0	0	disp		6
Program-counter relative	BRN (BF)	d:8	4	1	disp				4
		d:16	5	8	1	0	disp		6
Program-counter relative	BHI	d:8	4	2	disp				4
		d:16	5	8	2	0	disp		6
Program-counter relative	BLS	d:8	4	3	disp				4
		d:16	5	8	3	0	disp		6
Program-counter relative	Bcc (BHS)	d:8	4	4	disp				4
		d:16	5	8	4	0	disp		6
Program-counter relative	BCS (BLO)	d:8	4	5	disp				4
		d:16	5	8	5	0	disp		6
Program-counter relative	BNE	d:8	4	6	disp				4
		d:16	5	8	6	0	disp		6
Program-counter relative	BEQ	d:8	4	7	disp				4
		d:16	5	8	7	0	disp		6
Program-counter relative	BVC	d:8	4	8	disp				4
		d:16	5	8	8	0	disp		6
Program-counter relative	BVS	d:8	4	9	disp				4
		d:16	5	8	9	0	disp		6
Program-counter relative	BPL	d:8	4	A	disp				4
		d:16	5	8	A	0	disp		6
Program-counter relative	BMI	d:8	4	B	disp				4
		d:16	5	8	B	0	disp		6
Program-counter relative	BGE	d:8	4	C	disp				4
		d:16	5	8	C	0	disp		6
Program-counter relative	BLT	d:8	4	D	disp				4
		d:16	5	8	D	0	disp		6
Program-counter relative	BGT	d:8	4	E	disp				4
		d:16	5	8	E	0	disp		6
Program-counter relative	BLE	d:8	4	F	disp				4
		d:16	5	8	F	0	disp		6

Notes

1. The branch destination address must be even.
2. In machine language BRA, BRN, BCC, and BCS are identical to BT, BF, BHS, and BLO, respectively. The number of execution states for BRn (BF) is the same as for two NOP instructions.

2.2.8 BCLR

BCLR (Bit CLeaR) **Bit Clear**

Operation

0 → (<bit No.> of <EAd>)

Assembly-Language Format

BCLR #xx:3, <EAd>

BCLR Rn, <EAd>

Operand Size

Byte

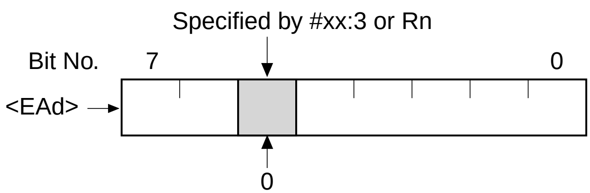
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction clears a specified bit in the destination operand to 0. The bit number can be specified by 3-bit immediate data, or by the lower three bits of a general register (Rn). The specified bit is not tested. The condition-code flags are not altered.



Available Registers

Rd: R0L to R7L, R0H to R7H
Rn: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

BCLR

BCLR (Bit CLeaR) — Bit Clear

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	BCLR	#xx:3, Rd	7 2	0 IMM rd			2
Register indirect	BCLR	#xx:3, @ERd	7 D	0 erd 0	7 2	0 IMM 0	8
Absolute address	BCLR	#xx:3, @aa:8	7 F	abs	7 2	0 IMM 0	8
Register direct	BCLR	Rn, Rd	6 2	rn rd			2
Register indirect	BCLR	Rn, @ERd	7 D	0 erd 0	6 2	rn 0	8
Absolute address	BCLR	Rn, @aa:8	7 F	abs	6 2	rn 0	8

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.9 BIAND

BIAND (Bit Invert AND) **Bit Logical AND**

Operation

C ∧ [¬ (<bit No.> of <EAd>)] → C

Assembly-Language Format

BIAND #xx:3, <EAd>

Operand Size

Byte

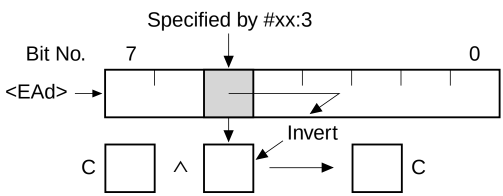
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Stores the result of the operation.

Description

This instruction ANDs the inverse of a specified bit in the destination operand with the carry bit and stores the result in the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	BIAND	#xx:3.Rd	7 6	1 IMM rd			2
Register indirect	BIAND	#xx:3.@ERd	7 C	0 erd 0	7 6	1 IMM 0	6
Absolute address	BIAND	#xx:3.@aa:8	7 E	abs	7 6	1 IMM 0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.10 BILD

BILD (Bit Invert LoaD) **Bit Load**

Operation

¬ (<bit No.> of <EAd>) → C

Assembly-Language Format

`BILD` #xx:3, <EAd>

Operand Size

Byte

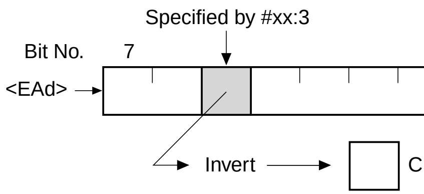
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Loaded with the inverse of the specified bit.

Description

This instruction loads the inverse of a specified bit from the destination operand into the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	BILD	#xx:3.Rd	7 7	1 IMM rd			2
Register indirect	BILD	#xx:3.@ERd	7 C	0 erd 0	7 7	1 IMM 0	6
Absolute address	BILD	#xx:3.@aa:8	7 E	abs	7 7	1 IMM 0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.11 BIOR

BIOR (Bit Invert inclusive OR) **Bit Logical OR**

Operation

C ∨ [¬ (<bit No.> of <EAd>)] → C

Assembly-Language Format

`BIOR` #xx:3, <EAd>

Operand Size

Byte

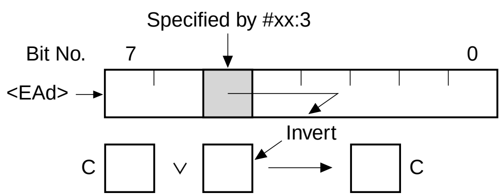
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Stores the result of the operation.

Description

This instruction ORs the inverse of a specified bit in the destination operand with the carry bit and stores the result in the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	BIOR	#xx:3.Rd	7	4	1 IMM	rd					2
Register indirect	BIOR	#xx:3.@ERd	7	C	0 erd	0	7	4	1 IMM	0	6
Absolute address	BIOR	#xx:3.@aa:8	7	E	abs		7	4	1 IMM	0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.12 BIST

BIST (Bit Invert STore) **Bit Store**

Operation

¬ C → (<bit No.> of <EAd>)

Assembly-Language Format

`BIST` #xx:3, <EAd>

Operand Size

Byte

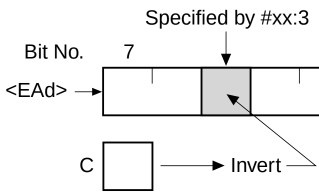
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction stores the inverse of the carry bit in a specified bit location in the destination operand. The bit number is specified by 3-bit immediate data. Other bits in the destination operand remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	BIST	#xx:3,Rd	6 7	1 IMM rd			2
Register indirect	BIST	#xx:3,@ERd	7 D	0 erd 0	6 7	1 IMM 0	8
Absolute address	BIST	#xx:3,@aa:8	7 F	abs	6 7	1 IMM 0	8

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.13 BIXOR

BIXOR (Bit Invert eXclusive OR) **Bit Exclusive Logical OR**

Operation

C ⊕ [¬ (<bit No.> of <EAd>)] → C

Assembly-Language Format

BIXOR #xx:3, <EAd>

Operand Size

Byte

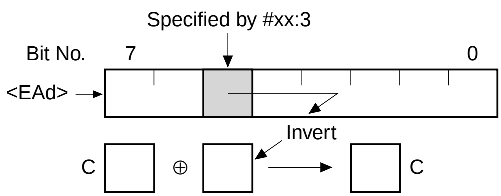
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Stores the result of the operation.

Description

This instruction exclusively ORs the inverse of a specified bit in the destination operand with the carry bit and stores the result in the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format										No. of States
			1st byte		2nd byte			3rd byte		4th byte			
Register direct	BIXOR	#xx:3,Rd	7	5	1	IMM	rd						2
Register indirect	BIXOR	#xx:3,@ERd	7	C	0	erd	0	7	5	1	IMM	0	6
Absolute address	BIXOR	#xx:3,@aa:8	7	E	abs			7	5	1	IMM	0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.14 BLD

BLD (Bit LoaD) **Bit Load**

Operation

(<Bit No.> of <EAd>) → C

Assembly-Language Format

`BLD` #xx:3, <EAd>

Operand Size

Byte

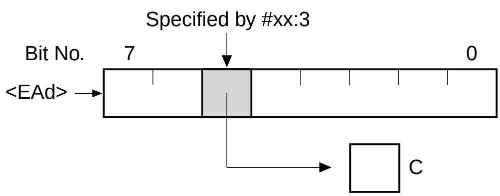
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Loaded from the specified bit.

Description

This instruction loads a specified bit from the destination operand into the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format 1st byte	Instruction Format 2nd byte	Instruction Format 3rd byte	Instruction Format 4th byte	No. of States
Register direct	BLD	#xx:3,Rd	7 7	0 IMM rd			2
Register indirect	BLD	#xx:3,@ERd	7 C	0 erd 0	7 7	0 IMM 0	6
Absolute address	BLD	#xx:3,@aa:8	7 E	abs	7 7	0 IMM 0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.15 BNOT

BNOT (Bit NOT) **Bit NOT**

Operation

¬ (<bit No.> of <EAd>) → (<bit No.> of <EAd>)

Assembly-Language Format

`BNOT` #xx:3, <EAd>
`BNOT` Rn, <EAd>

Operand Size

Byte

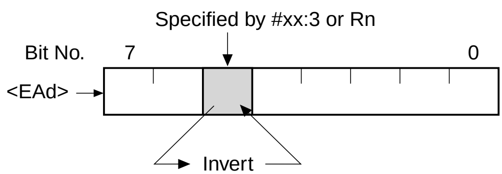
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction inverts a specified bit in the destination operand. The bit number is specified by 3-bit immediate data or by the lower 3 bits of a general register. The specified bit is not tested. The condition code remains unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
Rn: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

BNOT

BNOT (Bit NOT) Bit NOT

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	BNOT	#xx:3, Rd	7	1	0 IMM	rd					2
Register indirect	BNOT	#xx:3, @ERd	7	D	0 erd	0	7	1	0 IMM	0	8
Absolute address	BNOT	#xx:3, @aa:8	7	F	abs		7	1	0 IMM	0	8
Register direct	BNOT	Rn, Rd	6	1	rn	rd					2
Register indirect	BNOT	Rn, @ERd	7	D	0 erd	0	6	1	rn	0	8
Absolute address	BNOT	Rn, @aa:8	7	F	abs		6	1	rn	0	8

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.16 BOR

BOR (bit inclusive OR) **Bit Logical OR**

Operation

C ∨ [(<bit No.> of <EAd>)] → C

Assembly-Language Format

`BOR` #xx:3, <EAd>

Operand Size

Byte

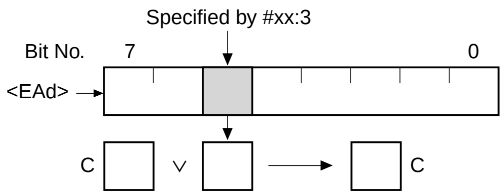
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Stores the result of the operation.

Description

This instruction ORs a specified bit in the destination operand with the carry bit and stores the result in the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format 1st byte	2nd byte	3rd byte	4th byte	No. of States
Register direct	BOR	#xx:3,Rd	7 4	0 IMM rd			2
Register indirect	BOR	#xx:3,@ERd	7 C	0 erd 0	7 4	0 IMM 0	6
Absolute address	BOR	#xx:3,@aa:8	7 E	abs	7 4	0 IMM 0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.17 BSET

BSET (Bit SET) **Bit Set**

Operation

1 → (<bit No.> of <EAd>)

Assembly-Language Format

BSET #xx:3, <EAd>
BSET Rn, <EAd>

Operand Size

Byte

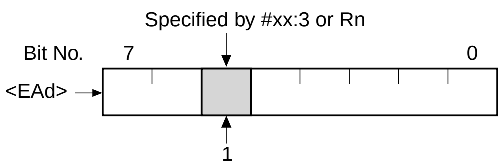
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction sets a specified bit in the destination operand to 1. The bit number can be specified by 3-bit immediate data, or by the lower three bits of a general register. The specified bit is not tested. The condition code flags are not altered.



Available Registers

Rd: R0L to R7L, R0H to R7H
Rn: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

BSET

BSET (Bit SET) — Bit Set

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	BSET	#xx:3, Rd	7	0	0 IMM	rd					2
Register indirect	BSET	#xx:3, @ERd	7	D	0 erd	0	7	0	0 IMM	0	8
Absolute address	BSET	#xx:3, @aa:8	7	F	abs		7	0	0 IMM	0	8
Register direct	BSET	Rn, Rd	6	0	rn	rd					2
Register indirect	BSET	Rn, @ERd	7	D	0 erd	0	6	0	rn	0	8
Absolute address	BSET	Rn, @aa:8	7	F	abs		6	0	rn	0	8

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.
<EAd> is byte data in a register or on memory.

2.2.18 BSR

BSR (Branch to SubRoutine) **Branch to Subroutine**

Operation

PC → @–SP
PC + disp → PC

Assembly-Language Format

BSR disp

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction branches to a subroutine at a specified address. It pushes the program counter (PC) value onto the stack as a restart address, then adds a specified displacement to the PC value and branches to the resulting address. The PC value pushed onto the stack is the address of the instruction following the BSR instruction. The displacement is a signed 8-bit or 16-bit value, so the possible branching range is –126 to +128 bytes or –32766 to +32768 bytes from the address of the BSR instruction.

Operand Format and Number of States Required for Execution

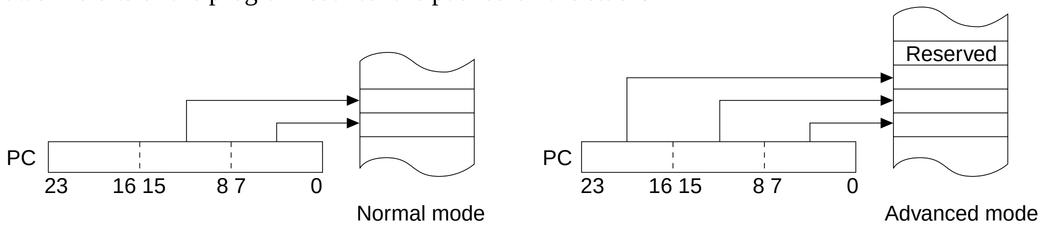
Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States	
			1st byte		2nd byte		3rd byte	4th byte	Normal	Advanced
Program-counter relative	BSR	d:8	5	5	disp				6	8
		d:16	5	C	0	0	disp		8	10

BSR

BSR (Branch to SubRoutine) **Branch to Subroutine**

Notes

The stack structure differs between normal mode and advanced mode. In normal mode only the lower 16 bits of the program counter are pushed on the stack.



The branch address must be even.

2.2.19 BST

BST (Bit STore) **Bit Store**

Operation

C → (<bit No.> of <EAd>)

Assembly-Language Format

BST #xx:3, <EAd>

Operand Size

Byte

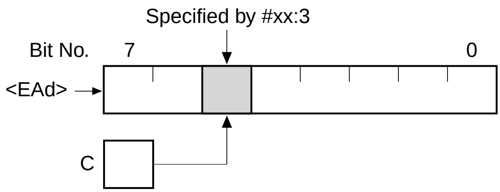
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction stores the carry bit in a specified bit location in the destination operand. The bit number is specified by 3-bit immediate data. Other bits in the destination operand remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	BST	#xx:3,Rd	6 7	0 IMM rd			2
Register indirect	BST	#xx:3,@ERd	7 D	0 erd 0	6 7	0 IMM 0	8
Absolute address	BST	#xx:3,@aa:8	7 F	abs	6 7	0 IMM 0	8

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.20 BTST

BTST (Bit TeST) **Bit Test**

Operation

¬ (<Bit No.> of <EAd>) → Z

Assembly-Language Format

BTST #xx:3, <EAd>
BTST Rn, <EAd>

Operand Size

Byte

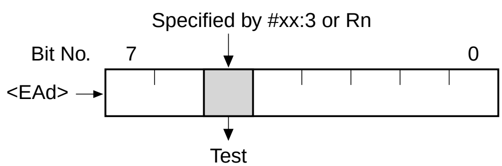
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	↕	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Set to 1 if the specified bit is zero; otherwise cleared to 0.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction tests a specified bit in the destination operand and sets or clears the Z flag according to the result. The bit number can be specified by 3-bit immediate data, or by the lower three bits of a general register. The destination operand remains unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
Rn: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

BTST

BTST (Bit TeST) **Bit Test**

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	BTST	#xx:3, Rd	7	3	0 IMM	rd					2
Register indirect	BTST	#xx:3, @ERd	7	C	0 erd	0	7	3	0 IMM	0	6
Absolute address	BTST	#xx:3, @aa:8	7	E	abs		7	3	0 IMM	0	6
Register direct	BTST	Rn, Rd	6	3	rn	rd					2
Register indirect	BTST	Rn, @ERd	7	C	0 erd	0	6	3	rn	0	6
Absolute address	BTST	Rn, @aa:8	7	E	abs		6	3	rn	0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.21 BXOR

BXOR (Bit eXclusive OR) **Bit Exclusive Logical OR**

Operation

C ⊕ (<bit No.> of <EAd>) → C

Assembly-Language Format

BXOR #xx:3, <EAd>

Operand Size

Byte

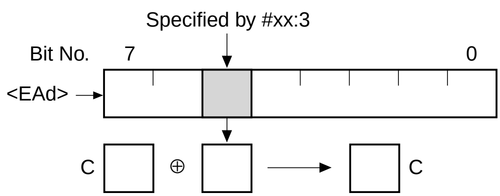
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	↕

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Stores the result of the operation.

Description

This instruction exclusively ORs a specified bit in the destination operand with the carry bit and stores the result in the carry bit. The bit number is specified by 3-bit immediate data. The destination operand contents remain unchanged.



Available Registers

Rd: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode*	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	BXOR	#xx:3,Rd	7 5	0 IMM rd			2
Register indirect	BXOR	#xx:3,@ERd	7 C	0 erd 0	7 5	0 IMM 0	6
Absolute address	BXOR	#xx:3,@aa:8	7 E	abs	7 5	0 IMM 0	6

Note: * The addressing mode is the addressing mode of the destination operand <EAd>.

Notes

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.22 (1) CMP (B)

CMP (CoMPare) **Compare**

Operation

Rd – (EAs), set or clear CCR

Assembly-Language Format

`CMP.B` <EAs>, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

Description

This instruction subtracts the source operand from the contents of an 8-bit register Rd (destination register) and sets or clears the CCR bits according to the result. The destination register contents remain unchanged.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	CMP.B	#xx:8, Rd	A	rd	IMM				2
Register direct	CMP.B	Rs, Rd	1	C	rs	rd			2

Notes

2.2.22 (2) CMP (W)

CMP (CoMPare) **Compare**

Operation

Rd – (EAs), set CCR

Assembly-Language Format

`CMP.W` <EAs>, Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.

Description

This instruction subtracts the source operand from the contents of a 16-bit register Rd (destination register) and sets or clears the CCR bits according to the result. The contents of the 16-bit register Rd remain unchanged.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	CMP.W	#xx:16, Rd	7	9	2	rd	IMM		4
Register direct	CMP.W	Rs, Rd	1	D	rs	rd			2

Notes

2.2.22 (3) CMP (L)

CMP (CoMPare) **Compare**

Operation

ERd – (EAs), set CCR

Assembly-Language Format

`CMP.L` <EAs>, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

I: Previous value remains unchanged.
H: Set to 1 if there is a borrow at bit 27; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.

Description

This instruction subtracts the source operand from the contents of a 32-bit register ERd (destination register) and sets or clears the CCR bits according to the result. The contents of the 32-bit register ERd remain unchanged.

Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	
Immediate	CMP.L	#xx:32, ERd	7 A	2 0 erd	IMM				6
Register direct	CMP.L	ERs, ERd	1 F	1 ers 0 erd					2

Notes

2.2.23 DAA

DAA (Decimal Adjust Add) **Decimal Adjust**

Operation

Rd (decimal adjust) → Rd

Assembly-Language Format

DAA Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	*	—	↕	↕	*	↕

H: Undetermined (no guaranteed value).
N: Set to 1 if the adjusted result is negative; otherwise cleared to 0.
Z: Set to 1 if the adjusted result is zero; otherwise cleared to 0.
V: Undetermined (no guaranteed value).
C: Set to 1 if there is a carry at bit 7; otherwise left unchanged.

Description

Given that the result of an addition operation performed by an ADD.B or ADDX instruction on 4-bit BCD data is contained in an 8-bit register Rd (destination register) and the carry and half-carry flags, the DAA instruction adjusts the general register contents by adding H'00, H'06, H'60, or H'66 according to the table below.

C Flag before Adjustment	Upper 4 Bits before Adjustment	H Flag before Adjustment	Lower 4 Bits before Adjustment	Value Added (hexadecimal)	C Flag after Adjustment
0	0 to 9	0	0 to 9	00	0
0	0 to 8	0	A to F	06	0
0	0 to 9	1	0 to 3	06	0
0	A to F	0	0 to 9	60	1
0	9 to F	0	A to F	66	1
0	A to F	1	0 to 3	66	1
1	1 to 2	0	0 to 9	60	1
1	1 to 2	0	A to F	66	1
1	1 to 3	1	0 to 3	66	1

Available Registers

Rd: R0L to R7L, R0H to R7H

DAA

DAA (Decimal Adjust Add) Decimal Adjust

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	DAA	Rd	0	F	0	rd			2

Notes

Valid results (8-bit register Rd contents and C, V, Z, N, and H flags) are not assured if this instruction is executed under conditions other than those described above.

2.2.24 DAS

DAS (Decimal Adjust Subtract) **Decimal Adjust**

Operation

Rd (decimal adjust) → Rd

Assembly-Language Format

DAS Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	*	—	↕	↕	*	—

H: Undetermined (no guaranteed value).
N: Set to 1 if the adjusted result is negative; otherwise cleared to 0.
Z: Set to 1 if the adjusted result is zero; otherwise cleared to 0.
V: Undetermined (no guaranteed value).
C: Previous value remains unchanged.

Description

Given that the result of a subtraction operation performed by a SUB.B, SUBX.B, or NEG.B instruction on 4-bit BCD data is contained in an 8-bit register Rd (destination register) and the carry and half-carry flags, the DAS instruction adjusts the general register contents by adding H'00, H'FA, H'A0, or H'9A according to the table below.

C Flag before Adjustment	Upper 4 Bits before Adjustment	H Flag before Adjustment	Lower 4 Bits before Adjustment	Value Added (hexadecimal)	C Flag after Adjustment
0	0 to 9	0	0 to 9	00	0
0	0 to 8	1	6 to F	FA	0
1	7 to F	0	0 to 9	A0	1
1	6 to F	1	6 to F	9A	1

Available Registers

Rd: R0L to R7L, R0H to R7H

DAS

DAS (Decimal Adjust Subtract) **Decimal Adjust**

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DAS	Rd	1 F	0 rd			2

Notes

Valid results (8-bit register Rd contents and C, V, Z, N, and H flags) are not assured if this instruction is executed under conditions other than those described above.

2.2.25 (1) DEC (B)

DEC (DECrement) **Decrement**

Operation

Rd – 1 → Rd

Assembly-Language Format

DEC.B Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs (the previous value in Rd was H'80); otherwise cleared to 0.
C: Previous value remains unchanged.

Description

This instruction decrements an 8-bit register Rd (destination register) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DEC.B	Rd	1 A	0 rd			2

Notes

An overflow is caused by the operation H'80 – 1 → H'7F.

2.2.25 (2) DEC (W)

DEC (DECrement) **Decrement**

Operation

Rd – 1 → Rd
Rd – 2 → Rd

Assembly-Language Format

`DEC.W #1,` Rd
`DEC.W #2,` Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs (the previous value in Rd was H'8000); otherwise cleared to 0.
C: Previous value remains unchanged.

Description

This instruction subtracts the immediate value 1 or 2 from the contents of a 16-bit register Rd (destination register) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	DEC.W	#1, Rd	1	B	5	rd			2
Register direct	DEC.W	#2, Rd	1	B	D	rd			2

Notes

An overflow is caused by the operations H'8000 – 1 → H'7FFF, H'8000 – 2 → H'7FFE, and H'8001 – 2 → H'7FFF.

2.2.25 (3) DEC (L)

DEC (DECrement) **Decrement**

Operation

ERd – 1 → ERd
ERd – 2 → ERd

Assembly-Language Format

`DEC.L #1,` ERd
`DEC.L #2,` ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Previous value remains unchanged.

Description

This instruction subtracts the immediate value 1 or 2 from the contents of a 32-bit register ERd (destination register) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DEC.L	#1, ERd	1 B	7 0 erd			2
Register direct	DEC.L	#2, ERd	1 B	F 0 erd			2

Notes

An overflow is caused by the operations H'80000000 – 1 → H'7FFFFFFF, H'80000000 – 2 → H'7FFFFFFE, and H'80000001 – 2 → H'7FFFFFFF.

2.2.26 (1) DIVXS (B)

DIVXS (DIVide eXtend as Signed) **Divide Signed**

Operation

Rd ÷ Rs → Rd

Assembly-Language Format

`DIVXS.B` Rs, Rd

Operand Size

Byte

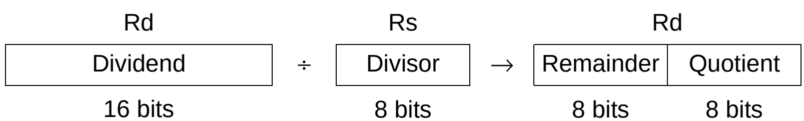
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	—	—

H: Previous value remains unchanged.
N: Set to 1 if the quotient is negative; otherwise cleared to 0.
Z: Set to 1 if the divisor is zero; otherwise cleared to 0.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction divides the contents of a 16-bit register Rd (destination register) by the contents of an 8-bit register Rs (source register) and stores the result in the 16-bit register Rd. The division is signed. The operation performed is 16 bits ÷ 8 bits → 8-bit quotient and 8-bit remainder. The quotient is placed in the lower 8 bits of Rd. The remainder is placed in the upper 8 bits of Rd.



Valid results are not assured if division by zero is attempted or an overflow occurs. For information on avoiding overflow, see DIVXS Instruction, Zero Divide, and Overflow.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0L to R7L, R0H to R7H

DIVXS (B)

DIVXS (DIVide eXtend as Signed) — Divide Signed

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	DIVXS.B	Rs, Rd	0	1	D	0	5	1	rs	rd	16

Notes

The N flag is set to 1 if the dividend and divisor have different signs, and cleared to 0 if they have the same sign. The N flag may therefore be set to 1 when the quotient is zero.

2.2.26 (2) DIVXS (W)

DIVXS (DIVide eXtend as Signed) **Divide Signed**

Operation

ERd ÷ Rs → ERd

Assembly-Language Format

`DIVXS.W` Rs, ERd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	—	—

H: Previous value remains unchanged.
N: Set to 1 if the quotient is negative; otherwise cleared to 0.
Z: Set to 1 if the divisor is zero; otherwise cleared to 0.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction divides the contents of a 32-bit register ERd (destination register) by the contents of a 16-bit register Rs (source register) and stores the result in the 32-bit register ERd. The division is signed. The operation performed is 32 bits ÷ 16 bits → 16-bit quotient and 16-bit remainder. The quotient is placed in the lower 16 bits (Rd) of the 32-bit register ERd. The remainder is placed in the upper 16 bits (Ed).

ERd		Rs		ERd	
Dividend	÷	Divisor	→	Remainder	Quotient
32 bits		16 bits		16 bits	16 bits

Valid results are not assured if division by zero is attempted or an overflow occurs. For information on avoiding overflow, see DIVXS Instruction, Zero Divide, and Overflow.

Available Registers

ERd: ER0 to ER7
Rs: R0 to R7, E0 to E7

DIVXS (W)

DIVXS (DIVide eXtend as Signed) — Divide Signed

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format									No. of States
			1st byte		2nd byte		3rd byte		4th byte			
Register direct	DIVXS.W	Rs, ERd	0	1	D	0	5	3	rs	0	erd	24

Notes

The N flag is set to 1 if the dividend and divisor have different signs, and cleared to 0 if they have the same sign. The N flag may therefore be set to 1 when the quotient is zero.

2.2.26 (3) DIVXS

DIVXS (DIVide eXtend as Signed) **Divide Signed**

DIVXS instruction, Division by Zero, and Overflow

Since the DIVXS instruction does not detect division by zero or overflow, applications should detect and handle division by zero and overflow using techniques similar to those used in the following program.

1. Programming solution for DIVXS.B R0L, R1

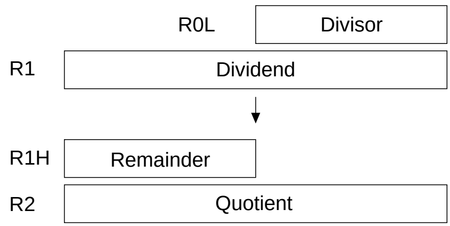
Example 1: Convert dividend and divisor to non-negative numbers, then use DIVXU programming solution for zero divide and overflow

```
      MOV.B     R0L, R0L     ; Test divisor
      BEQ       ZERODIV      ; Branch to ZERODIV if R0L = 0
      ANDC      #AF, CCR     ; Clear CCR user bits (bits 6 and 4) to 0
      BPL       L1           ; Branch to L1 if N flag = 0 (positive divisor)
      NEG.B     R0L          ; Take 2's complement of R0L to make sign positive
      ORC       #10, CCR     ; Set CCR bit 4 to 1
L1:   MOV.W     R1.R1        ; Test dividend
      BPL       L2           ; Branch to L2 if N flag = 0 (positive dividend)
      NEG.W     R1           ; Take 2's complement of R1 to make sign positive
      XORC      #50, CCR     ; Invert CCR bits 6 and 4
L2:   MOV.B     R1H, R2L     ;
      EXTU.W    R2           ;
      DIVXU.B   R0L, R2      ; Use DIVXU.B instruction to divide non-negative dividend
      MOV.B     R2H, R1H     ; by positive divisor
      DIVXU.B   R0L, R1      ; 16 bits ÷ 8 bits → quotient (16 bits) and remainder (8 bits)
      MOV.B     R2L, R2H     ; (See DIVXU Instruction, Zero Divide, and Overflow)
      MOV.B     R1L, R2L     ;
      STC       CCR, R1L     ; Copy CCR contents to R1L
      BTST      #6, R1L      ; Test CCR bit 6
      BEQ       L3           ; Branch to L3 if bit 6 = 1
      NEG.B     R1H          ; Take 2's complement of R1H to make sign of remainder negative
L3:   BTST      #4, R1L      ; Test CCR bit 4
      BEQ       L4           ; Branch to L4 if bit 4 = 1
      NEG.W     R2           ; Take 2's complement of R2 to make sign of quotient negative
L4:   RTS
ZERODIV:                     ; Zero-divide handling routine
```

DIVXS

DIVXS (DIVide eXtend as Signed) — Divide Signed

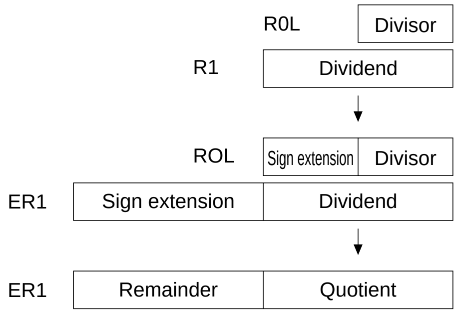
This program leaves a 16-bit quotient in R2 and an 8-bit remainder in R1H.



Example 2: Sign extend the 8-bit divisor to 16 bits, sign extend the 16-bit dividend to 32 bits, and then use DIVXS to divide

```
    EXTS.W    R0
    BEQ       ZERODIV
    EXTS.L    ER1
    DIVXS.L   R0,ER1
    RTS
ZERODIV:
```

This program leaves the 16-bit quotient in R1 and the 8-bit remainder in E1 (in a 16-bit sign extended format).



DIVXS

DIVXS (DIVide eXtend as Signed) **Divide Signed**

2. Programming solution for DIVXS.W R0, ER1

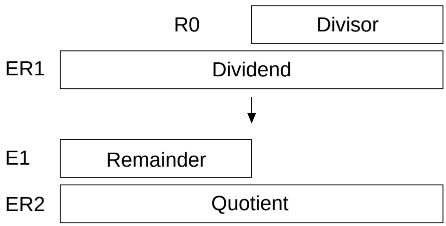
Example: Convert dividend and divisor to non-negative numbers, then use DIVXU programming solution for zero divide and overflow

```
      MOV.W      R0, R0      ;  Test divisor
      BEQ        ZERODIV     ;  Branch to ZERODIV if R0 = 0
      ANDC       #AF, CCR    ;  Clear CCR user bits (bits 6 and 4) to 0
      BPL        L1          ;  Branch to L1 if N flag = 0 (positive divisor)
      NEG.W      R0          ;  Take 2's complement of R0 to make sign positive
      ORC        #10, CCR    ;  Set CCR bit 4 to 1
L1:   MOV.L      ER1,ER1     ;  Test dividend
      BPL        L2          ;  Branch to L2 if N flag = 0 (positive dividend)
      NEG.L      ER1         ;  Take 2's complement of ER1 to make sign positive
      XORC       #50,CCR     ;  Invert CCR bits 6 and 4
L2:   MOV.W      E1, R2      ;
      EXTU.L     ER2         ;
      DIVXU.W    R0, E2      ;     Use DIVXU.W instruction to divide non-negative dividend
      MOV.W      E2, R1      ;     by positive divisor
      DIVXU.W    R0, ER1     ;     32 bits ÷ 16 bits → quotient (32 bits) and remainder
      MOV.W      R2, E2            (16 bits)
      MOV.W      R1, R2            (See DIVXU Instruction, Zero Divide, and Overflow)
      STC        CCR, R1L    ;  Copy CCR contents to R1L
      BTST       #6, R1L     ;  Test CCR bit 6
      BEQ        L3          ;  Branch to L3 if bit 6 = 1
      NEG.W      E1          ;  Take 2's complement of E1 to make sign of remainder negative
L3:   BTST       #4, R1L     ;  Test CCR bit 4
      BEQ        L4          ;  Branch to L4 if bit 4 = 1
      NEG.L      ER2         ;  Take 2's complement of ER2 to make sign of quotient negative
L4:   RTS
ZERODIV:                     ;  Zero-divide handling routine
```

DIVXS

DIVXS (DIVide eXtend as Signed) Divide Signed

This program leaves a 32-bit quotient in ER2 and a 16-bit remainder in E1.



The preceding two examples flag the status of the divisor and dividend in the UI and U bits in the CCR, and modify the sign of the quotient and remainder in the unsigned division result of the DIVXU instruction as shown next.

UI	U	Divisor	Dividend	Remainder	Quotient	Sign Modification
0	0	Positive	Positive	Positive	Positive	No sign modification
0	1	Negative	Positive	Positive	Negative	Sign of quotient is reversed
1	0	Negative	Negative	Negative	Positive	Sign of remainder is reversed
1	1	Positive	Negative	Negative	Negative	Signs of quotient and remainder are both reversed

2.2.27 (1) DIVXU (B)

DIVXU (DIVide eXtend as Unsigned) **Divide**

Operation

Rd ÷ Rs → Rd

Assembly-Language Format

`DIVXU.B` Rs, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	—	—

H: Previous value remains unchanged.
N: Set to 1 if the divisor is negative; otherwise cleared to 0.
Z: Set to 1 if the divisor is zero; otherwise cleared to 0.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction divides the contents of a 16-bit register Rd (destination register) by the contents of an 8-bit register Rs (source register) and stores the result in the 16-bit register Rd. The division is unsigned. The operation performed is 16 bits ÷ 8 bits → 8-bit quotient and 8-bit remainder. The quotient is placed in the lower 8 bits of Rd. The remainder is placed in the upper 8 bits of Rd.

Rd		Rs		Rd	
Dividend	÷	Divisor	→	Remainder	Quotient
16 bits		8 bits		8 bits	8 bits

Valid results are not assured if division by zero is attempted or an overflow occurs. For information on avoiding overflow, see DIVXU Instruction, Zero Divide, and Overflow.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	DIVXU.B	Rs, Rd	5	1	rs	rd			14

Notes

2.2.27 (2) DIVXU (W)

DIVXU (DIVide eXtend as Unsigned) **Divide**

Operation

ERd ÷ Rs → ERd

Assembly-Language Format

`DIVXU.W` Rs, ERd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	—	—

H: Previous value remains unchanged.
N: Set to 1 if the divisor is negative; otherwise cleared to 0.
Z: Set to 1 if the divisor is zero; otherwise cleared to 0.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction divides the contents of a 32-bit register ERd (destination register) by the contents of a 16-bit register Rs (source register) and stores the result in the 32-bit register ERd. The division is unsigned. The operation performed is 32 bits ÷ 16 bits → 16-bit quotient and 16-bit remainder. The quotient is placed in the lower 16 bits (Rd) of the 32-bit register ERd. The remainder is placed in the upper 8 bits of (Ed).

ERd		Rs		ERd	
Dividend	÷	Divisor	→	Remainder	Quotient
32 bits		16 bits		16 bits	16 bits

Valid results are not assured if division by zero is attempted or an overflow occurs. For information on avoiding overflow, see DIVXU Instruction, Zero Divide, and Overflow.

Available Registers

ERd: ER0 to ER7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	DIVXU.W	Rs, ERd	5 3	rs 0 ERd			22

Notes

DIVXU

DIVXU (DIVide eXtend as Unsigned) — Divide

DIVXU Instruction, Zero Divide, and Overflow

Zero divide and overflow are not detected in the DIVXU instruction. A program like the following can detect zero divisors and avoid overflow.

1. Programming solutions for DIVXU.B R0L, R1

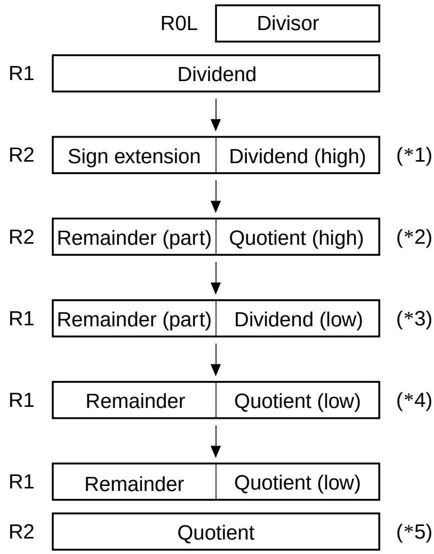
Example 1: Divide upper 8 bits and lower 8 bits of 16-bit dividend separately and obtain 16-bit quotient

```
        CMP.B      #0, R0L          ; R0L = 0? (Zero divisor?)
        BEQ        ZERODIV          ; Branch to ZERODIV if R0L = 0
        MOV.B      R1H,R2L          ;    Copy upper 8 bits of dividend to R2L and
        EXTU.W     R2 (*1).         ;    zero-extend to 16 bits
        DIVXU.B    R0L, R2 (*2)     ; Divide upper 8 bits of dividend
        MOV.B      R2H, R1H (*3)    ; R2H → R1H (store partial remainder in R1H)
        DIVXU.B    R0L, R1 (*4)     ; Divide lower 8 bits of dividend (including repeated
                                      division of upper 8 bits)
        MOV.B      R2L, R2H         ; Store upper part of quotient in R2H
        MOV.B      R1L, R2L (*5)    ; Store lower part of quotient in R2L
        RTS
ZERODIV:                            ; Zero-divide handling routine
```

DIVXU

DIVXU (DIVide eXtend as Unsigned) **Divide**

The resulting operation is 16 bits ÷ 8 bits → quotient (16 bits) and remainder (8 bits), and no overflow occurs. The 16-bit quotient is stored in R2, the 8-bit remainder in R1H.



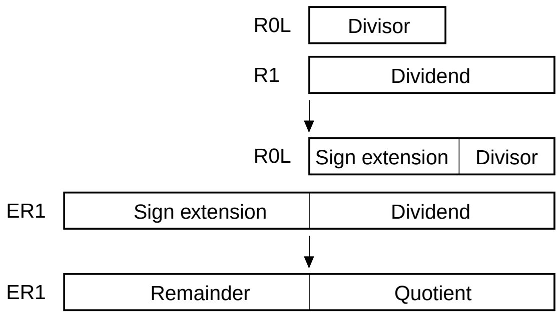
DIVXU

DIVXU (DIVide eXtend as Unsigned) **Divide**

Example 2: Zero-extend divisor from 8 to 16 bits and dividend from 16 to 32 bits before dividing

```
    EXTU.W      R0          ; Zero-extend 8-bit divisor to 16 bits
    BEQ         ZERODIV     ; Branch to ZERODIV if R0 = 0
    EXTU.L      ER1         ; Zero-extend 16-bit dividend to 32 bits
    EXTU.W      R0, ER1     ; Divide using DIVXU.W
    RTS
ZERODIV:                    ; Zero-divide handling routine
```

Instead of 16 bits ÷ 8 bits, the operation performed is 32 bits ÷ 16 bits → quotient (16 bits) and remainder (16 bits), and no overflow occurs. The 16-bit quotient is stored in R1 and the 8-bit remainder in the lower 8 bits of E1. The upper 8 bits of E1 are all 0.



DIVXU

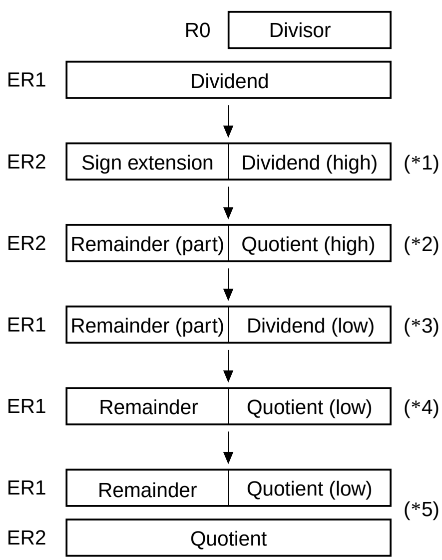
DIVXU (DIVide eXtend as Unsigned) **Divide**

2. Programming solution for DIVXU.W R0, ER1

Example 1: Divide upper 16 bits and lower 16 bits of 32-bit dividend separately and obtain 32-bit quotient

```
    MOV.W      R0, R0             ; R0 = 0? (Zero divisor?)
    BEQ        ZERODIV            ; Branch to ZERODIV if R0 = 0
    MOV.W      E1,E2              ; Copy upper 16 bits of dividend to R2 and
    EXTU.L     ER2       (*1)     ; zero-extend to 32 bits
    DIVXU.W    R0, ER2   (*2)     ; Divide upper 16 bits of dividend
    MOV.W      E2, E1    (*3)     ; E2 → E1 (store partial remainder in E1)
    DIVXU.W    R0, ER1   (*4)     ; Divide lower 16 bits of dividend (including repeated
                                    division of upper 16 bits)
    MOV.W      R2, E2             ; Store upper part of quotient in E2
    MOV.W      R1, R2    (*5)     ; Store lower part of quotient in R2
    RTS
ZERODIV:                          ; Zero-divide handling routine
```

The resulting operation is 32 bits ÷ 16 bits → quotient (32 bits) and remainder (16 bits), and no overflow occurs. The 32-bit quotient is stored in ER2, the 16-bit remainder in E1.



2.2.28 (1) EEPMOV (B)

EEPMOV (MOVe data to EEPROM) **Block Data Transfer**

Operation

if R4L ≠ 0 then
 repeat @ER5+ → @ER6+
 R4L – 1 → R4L
 until R4L = 0
else next;

Assembly-Language Format

EEPMOV.B

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction performs a block memory transfer. It moves data from the memory location specified in ER5 to the memory location specified in ER6, increments ER5 and ER6, decrements R4L, and repeats these operations until R4L reaches zero. Execution then proceeds to the next instruction. No interrupts are detected while the block transfer is in progress. When the EEPMOV instruction ends, R4L contains 0, and ER5 and ER6 contain the last transfer address + 1. The data transfer is performed a byte at a time, with R4L indicating the number of bytes to be transferred. The byte symbol in the assembly-language format designates the size of R4L (and limits the maximum number of bytes that can be transferred to 255).

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
—	EEPMOV.B		7	B	5	C	5	9	8	F	8+4n*

Note: * n is the initial value of R4L. Although n bytes of data are transferred, memory is accessed 2(n + 1) times, requiring 4(n + 1) states. (n = 0, 1, 2, …, 255).

Notes

This instruction first reads the memory locations indicated by ER5 and ER6, then performs the data transfer. The number of states required for execution differs from the H8/300 CPU.

2.2.28 (2) EEPMOV (W)

EEPMOV (MOVe data to EEPROM) **Block Data Transfer**

Operation

if R4 ≠ 0 then
    repeat   @ER5+ → @ER6+
              R4 – 1 → R4
    until R4 = 0
else next;

Assembly-Language Format

`EEPMOV.W`

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction performs a block memory transfer. It moves data from the memory location specified in ER5 to the memory location specified in ER6, increments ER5 and ER6, decrements R4, and repeats these operations until R4 reaches zero. Execution then proceeds to the next instruction. No interrupts except NMI are detected while the block transfer is in progress. When the EEPMOV instruction ends, R4 contains 0, and ER5 and ER6 contain the last transfer address + 1. The data transfer is performed a byte at a time, with R4 indicating the number of bytes to be transferred. The word symbol in the assembly-language format designates the size of R4 (allowing a maximum 65535 bytes to be transferred).

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
—	EEPMOV.W		7	B	D	4	5	9	8	F	8+4n

Note: n is the initial value of R4. Although n bytes of data are transferred, memory is accessed 2(n + 1) times, requiring 4(n + 1) states. (n = 0, 1, 2, …, 65535).

Notes

This instruction first reads memory at the addresses indicated by ER5 and ER6, then carries out the block data transfer.

EEPMOV (W)

EEPMOV (MOVe data to EEPROM) **Block Data Transfer**

EEPMOV.W Instruction and NMI Interrupt

If an NMI request occurs while the EEPMOV.W instruction is being executed, NMI interrupt exception handling is carried out at the end of the current read-write cycle. Register contents are then as follows:

ER5: address of the next byte to be transferred
ER6: destination address of the next byte
R4: number of bytes remaining to be transferred

The program counter value pushed on the stack in NMI interrupt exception handling is the address of the next instruction after the EEPMOV.W instruction. Programs should be coded as follows to allow for NMI interrupts during execution of the EEPMOV.W instruction.

Example:

```
L1: EEPMOV.W
    MOV.W      R4, R4
    BNE        L1
```

During execution of the EEPMOV.B instruction no interrupts are accepted, including NMI.

2.2.29 (1) EXTS (W)

EXTS (EXTend as Signed) **Sign Extension**

Operation

(<Bit 7> of Rd) → (<bits 15 to 8> of Rd>

Assembly-Language Format

EXTS.W Rd

Operand Size

Word

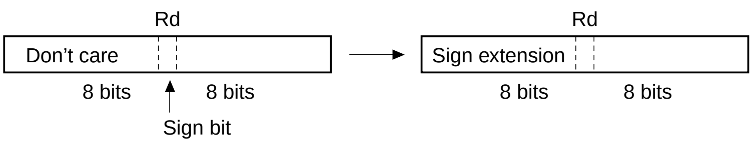
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction copies the sign of the lower 8 bits in a 16-bit register Rd in the upward direction (copies Rd bit 7 to bits 15 to 8) to extend the data to signed word data.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTS.W	Rd	1 7	D rd			2

Notes

2.2.29 (2) EXTS (L)

EXTS (EXTend as Signed) **Sign Extension**

Operation

(<Bit 15> of ERd) → (<bits 31 to 16> of ERd>)

Assembly-Language Format

`EXTS.L` ERd

Operand Size

Longword

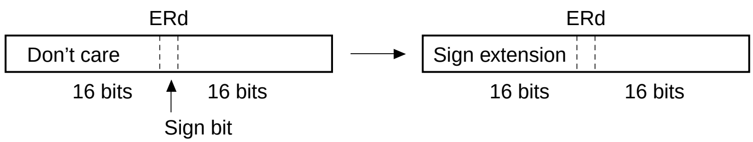
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction copies the sign of the lower 16 bits (general register Rd) in a 32-bit register ERd in the upward direction (copies ERd bit 15 to bits 31 to 16) to extend the data to signed longword data.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTS.L	ERd	1 7	F 0 erd			2

Notes

2.2.30 (1) EXTU (W)

EXTU (EXTend as Unsigned) **Zero Extension**

Operation

0 → (<bits 15 to 8> of Rd>)
Zero extend

Assembly-Language Format

`EXTU.W` Rd

Operand Size

Word

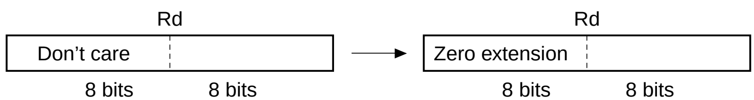
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	—

H: Previous value remains unchanged.
N: Always cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction extends the lower 8 bits in a 16-bit register Rd to word data by padding with zeros. That is, it clears the upper 8 bits of Rd (bits 15 to 8) to 0.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTU.W	Rd	1 7	5 rd			2

Notes

2.2.30 (2) EXTU (L)

EXTU (EXTend as Unsigned) **Zero Extension**

Operation

0 → (<bits 31 to 16> of ERd>)
Zero extend

Assembly-Language Format

`EXTU.L` ERd

Operand Size

Longword

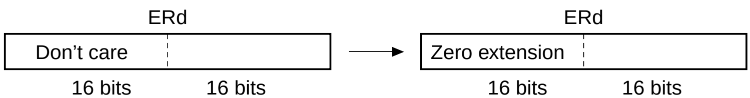
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	—

H: Previous value remains unchanged.
N: Always cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction extends the lower 16 bits (general register Rd) in a 32-bit register ERd to longword data by padding with zeros. That is, it clears the upper 16 bits of ERd (bits 31 to 16) to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	EXTU.L	ERd	1 7	7 0 erd			2

Notes

2.2.31 (1) INC (B)

INC (INCrement) **Increment**

Operation

Rd + 1 → Rd

Assembly-Language Format

`INC.B` Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Previous value remains unchanged.

Description

This instruction increments an 8-bit register Rd (destination register) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	INC.B	Rd	0 ⁞ A	0 ⁞ rd			2

Notes

An overflow is caused by the operation H'7F + 1 → H'80.

2.2.31 (2) INC (W)

INC (INCrement) **Increment**

Operation

Rd + 1 → Rd
Rd + 2 → Rd

Assembly-Language Format

`INC.W #1, Rd`
`INC.W #2, Rd`

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Previous value remains unchanged.

Description

This instruction adds the immediate value 1 or 2 to the contents of a 16-bit register Rd (destination register) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	INC.W	#1, Rd	0	B	5	rd			2
Register direct	INC.W	#2, Rd	0	B	D	rd			2

Notes

An overflow is caused by the operations H'7FFF + 1 → H'8000, H'7FFF + 2 → H'8001, and H'7FFE + 2 → H'8000.

2.2.31 (3) INC (L)

INC (INCrement) **Increment**

Operation

ERd + 1 → ERd
ERd + 2 → ERd

Assembly-Language Format

`INC.L #1, ERd`
`INC.L #2, ERd`

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Previous value remains unchanged.

Description

This instruction adds the immediate value 1 or 2 to the contents of a 32-bit register ERd (destination register) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	INC.L	#1, ERd	0	B	7	0 erd			2
Register direct	INC.L	#2, ERd	0	B	F	0 erd			2

Notes

An overflow is caused by the operations H'7FFFFFFF + 1 → H'80000000, H'7FFFFFFF + 2 → H'80000001, and H'7FFFFFFE + 2 → H'80000000.

2.2.32 JMP

JMP (JuMP) **Unconditional Branch**

Operation

Effective address → PC

Assembly-Language Format

`JMP` <EA>

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction branches unconditionally to a specified address

Available Registers

ERn: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st byte	2nd byte	3rd byte	4th byte	Normal	Advanced
Register indirect	JMP	@ERn	5 9	0 ern 0			4	
Absolute address	JMP	@aa:24	5 A	abs			6	
Memory indirect	JMP	@@aa:8	5 B	abs			8	10

Notes

The structure of the branch address and the number of states required for execution differ between normal mode and advanced mode.

The branch address must be even.

2.2.33 JSR

JSR (Jump to SubRoutine) **Jump to Subroutine**

Operation

PC → @–SP

Effective address → PC

Assembly-Language Format

JSR <EA>

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.

N: Previous value remains unchanged.

Z: Previous value remains unchanged.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction pushes the program counter on the stack as a return address, then branches to a specified effective address. The program counter value pushed on the stack is the address of the instruction following the JSR instruction.

Available Registers

ERn: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States	
			1st byte		2nd byte		3rd byte	4th byte	Normal	Advanced
Register indirect	JSR	@ERn	5	D	0 ern	0			6	8
Absolute address	JSR	@aa:24	5	E	abs				8	10
Memory indirect	JSR	@@aa:8	5	F	abs				8	12

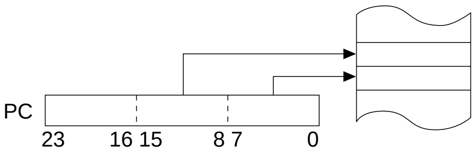
JSR

JSR (Jump to SubRoutine) **Jump to Subroutine**

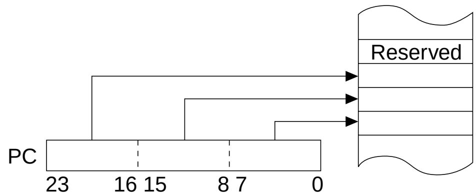
Notes

Note that the structures of the stack and branch addresses differ between normal and advanced mode. Only the lower 16 bits of the PC are saved in normal mode.

The branch address must be even.



Normal mode



Advanced mode

2.2.34 (1) LDC (B)

LDC (LoaD to Control register) **Load CCR**

Operation

(EAs) → CCR

Assembly-Language Format

`LDC.B` <EAs>, CCR

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

I: Loaded from the corresponding bit in the source operand.
H: Loaded from the corresponding bit in the source operand.
N: Loaded from the corresponding bit in the source operand.
Z: Loaded from the corresponding bit in the source operand.
V: Loaded from the corresponding bit in the source operand.
C: Loaded from the corresponding bit in the source operand.

Description

This instruction loads the source operand into the CCR.

Note that no interrupts, even NMI interrupts, will be accepted at the point that this instruction completes.

Available Registers

Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	LDC.B	#xx:8, CCR	0	7	IMM				2
Register direct	LDC.B	Rs, CCR	0	3	0	rs			2

Notes

2.2.34 (2) LDC (W)

LDC (LoaD to Control register) **Load CCR**

Operation

(EAs) $\rightarrow$ CCR

Assembly-Language Format

`LDC.W` <EAs>, CCR

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

I: Loaded from the corresponding bit in the source operand.
H: Loaded from the corresponding bit in the source operand.
N: Loaded from the corresponding bit in the source operand.
Z: Loaded from the corresponding bit in the source operand.
V: Loaded from the corresponding bit in the source operand.
C: Loaded from the corresponding bit in the source operand.

Description

This instruction loads the source operand contents into the condition-code register (CCR). Although CCR is a byte register, the source operand is word size. The contents of the even address are loaded into CCR.

No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

Available Registers

ERs: ER0 to ER7

LDC (W)

LDC (LoaD to Control register) — Load CCR

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	1st byte		2nd byte		3rd byte		4th byte		5th byte		6th byte		7th byte		8th byte	9th byte	10th byte	No. of States
Register indirect	LDC.W	@ERs,CCR	0	1	4	0	6	9	0:ers	0										6
Register indirect with displacement	LDC.W	@(d:16,ERs),CCR	0	1	4	0	6	F	0:ers	0	disp									8
	LDC.W	@(d:24,ERs),CCR	0	1	4	0	7	8	0:ers	0	6	B	2	0	0	0	disp			12
Register indirect with post-increment	LDC.W	@ERs+,CCR	0	1	4	0	6	D	0:ers	0										8
Absolute address	LDC.W	@aa:16,CCR	0	1	4	0	6	B	0	0	abs									8
	LDC.W	@aa:24,CCR	0	1	4	0	6	B	2	0	0	0	abs							10

Notes

2.2.35 (1) MOV (B)

MOV (MOVe data) **Move**

Operation

Rs → Rd

Assembly-Language Format

`MOV.B` Rs, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers one byte of data from an 8-bit register Rs to an 8-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MOV.B	Rs, Rd	0 ⋮ C	rs ⋮ rd			2

Notes

RENESAS

2.2.35 (2) MOV (W)

MOV (MOVe data) **Move**

Operation

Rs → Rd

Assembly-Language Format

`MOV.W` Rs, Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers one word of data from a 16-bit register Rs to a 16-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	MOV.W	Rs, Rd	0	D	rs	rd			2

Notes

2.2.35 (3) MOV (L)

MOV (MOVe data) **Move**

Operation

ERs → ERd

Assembly-Language Format

MOV.L ERs, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers one longword of data from a 32-bit register ERs to a 32-bit register ERd, tests the transferred data, and sets condition-code flags according to the result.

Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte				3rd byte	4th byte	
Register direct	MOV.L	ERs, ERd	0	F	1	ers	0	erd			2

Notes

2.2.35 (4) MOV (B)

MOV (MOVe data) **Move**

Operation

(EAs) → Rd

Assembly-Language Format

`MOV.B` <EAs>, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers the source operand contents to an 8-bit register Rs, tests the transferred data, and sets condition-code flags according to the result.

Available Registers

Rd: R0L to R7L, R0H to R7H
ERs: ER0 to ER7

MOV (B)

MOV (MOVe data) Move

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format: 1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	No. of States
Immediate	MOV.B	#xx:8,Rd	F rd	IMM							2
Register indirect	MOV.B	@ERs,Rd	6 8	0 ers rd							4
Register indirect with displacement	MOV.B	@(d:16,ERs),Rd	6 E	0 ers rd	disp						6
	MOV.B	@(d:24,ERs),Rd	7 8	0 ers 0	6 A	2 rd	0 0	disp			10
Register indirect with post-increment	MOV.B	@ERs+,Rd	6 C	0 ers rd							6
Absolute address	MOV.B	@aa:8,Rd	2 rd	abs							4
	MOV.B	@aa:16,Rd	6 A	0 rd	abs						6
	MOV.B	@aa:24,Rd	6 A	2 rd	0 0	abs					8

Notes

The MOV.B @ER7+, Rd instruction should never be used, because it leaves an odd value in the stack pointer (ER7). For details refer to section 3.3.2, Exception Processing, or to the hardware manual.

For the @aa:8 access range, refer to the relevant microcontroller hardware manual.

2.2.35 (5) MOV (W)

MOV (MOVe data) **Move**

Operation

(EAs) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Assembly-Language Format

`MOV.W` <EAs>, Rd

Operand Size

Word

Description

This instruction transfers the source operand contents to a 16-bit register Rd, tests the transferred data, and sets condition-code flags according to the result.

Available Registers

Rd: R0 to R7, E0 to E7
ERs: ER0 to ER7

MOV (W)

MOV (MOVe data) Move

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	No. of States
Immediate	MOV.W	#xx:16,Rd	7 9	0 rd	IMM						4
Register indirect	MOV.W	@ERs,Rd	6 9	0 ers rd							4
Register indirect with displacement	MOV.W	@(d:16,ERs),Rd	6 F	0 ers rd	disp						6
	MOV.W	@(d:24,ERs),Rd	7 8	0 ers 0	6 B	2 rd	0 0	disp			10
Register indirect with post-increment	MOV.W	@ERs+,Rd	6 D	0 ers rd							6
Absolute address	MOV.W	@aa:16,Rd	6 B	0 rd	abs						6
	MOV.W	@aa:24,Rd	6 B	2 rd	0 0	abs					8

Notes

1. The source operand <EAs> must be located at an even address.
2. In machine language, MOV.W @R7+, Rd is identical to POP.W Rd.

2.2.35 (6) MOV (L)

MOV (MOVe data) **Move**

Operation

(EAs) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Assembly-Language Format

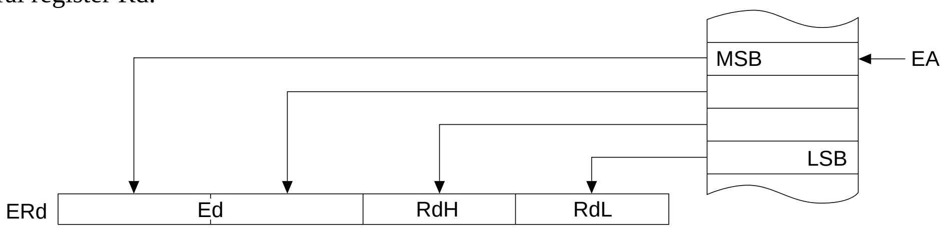
`MOV.L` <EAs>, ERd

Operand Size

Longword

Description

This instruction transfers the source operand contents to a specified 32-bit register (ERd), tests the transferred data, and sets condition-code flags according to the result. The first memory word located at the effective address is stored in extended register Ed. The next word is stored in general register Rd.



Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

MOV (L)

MOV (MOVe data) Move

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format: 1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte	No. of States
Immediate	MOV.L	#xx:32,Rd	7 A	0 0 ers	IMM								6
Register indirect	MOV.L	@ERs,ERd	0 1	0 0	6 9	0 ers 0 erd							8
Register indirect with displacement	MOV.L	@(d:16,ERs),ERd	0 1	0 0	6 F	0 ers 0 erd	disp						10
	MOV.L	@(d:24,ERs),ERd	0 1	0 0	7 8	0 ers 0	6 B	2 0 erd	0 0	disp			14
Register indirect with post-increment	MOV.L	@ERs+,ERd	0 1	0 0	6 D	0 ers 0 erd							10
Absolute address	MOV.L	@aa:16,ERd	0 1	0 0	6 B	0 0 erd	abs						10
	MOV.L	@aa:24,ERd	0 1	0 0	6 B	2 0 erd	0 0	abs					12

Notes

1. The source operand <EAs> must be located at an even address.
2. In machine language, MOV.L @ER7+, ERd is identical to POP.L ERd.

2.2.35 (7) MOV (B)

MOV (MOVe data) **Move**

Operation

Rs → (EAd)

Assembly-Language Format

`MOV.B` Rs, <EAd>

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers the contents of an 8-bit register Rs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result.

Available Registers

Rs: R0L to R7L, R0H to R7H
ERd: ER0 to ER7

MOV (B)

MOV (MOVe data) Move

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format: 1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	No. of States
Register indirect	MOV.B	Rs,@ERd	6 8	1 erd rs							4
Register indirect with displacement	MOV.B	Rs,@(d:16,ERd)	6 E	1 erd rs	disp						6
	MOV.B	Rs,@(d:24,ERd)	7 8	0 erd 0	6 A	A rs	0 0	disp			10
Register indirect with pre-decrement	MOV.B	Rs,@–ERd	6 C	1 erd rs							6
Absolute address	MOV.B	Rs,@aa:8	3 rs	abs							4
	MOV.B	Rs,@aa:16	6 A	8 rs	abs						6
	MOV.B	Rs,@aa:24	6 A	A rs	0 0	abs					8

Note: in the printed table, "disp" in the d:16 row spans the 3rd and 4th bytes, and in the d:24 row spans the 6th to 8th bytes; "abs" spans the 3rd and 4th bytes in the aa:16 row and the 4th to 6th bytes in the aa:24 row.

Notes

1. The MOV.B Rs, @–ER7 instruction should never be used, because it leaves an odd value in the stack pointer (ER7). For details refer to section 3.3.2, Exception Processing, or to the hardware manual.
2. Execution of MOV.B RnL, @–ERn or MOV.B RnH, @–ERn first decrements ERn by one, then transfers the designated part (RnL or RnH) of the resulting ERn value.

2.2.35 (8) MOV (W)

MOV (MOVe data) **Move**

Operation

Rs → (EAd)

Assembly-Language Format

`MOV.W` Rs, <EAd>

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers the contents of a 16-bit register Rs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result.

Available Registers

Rs: R0 to R7, E0 to E7
ERd: ER0 to ER7

MOV (W)

MOV (MOVe data) Move

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format 1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	No. of States
Register indirect	MOV.W	Rs,@ERd	6 9	1 erd rs							4
Register indirect with displacement	MOV.W	Rs,@(d:16,ERd)	6 F	1 erd rs	disp						6
	MOV.W	Rs,@(d:24,ERd)	7 8	0 erd 0	6 B	A rs	0 0	disp			10
Register indirect with post-increment	MOV.W	Rs,@–ERd	6 D	1 erd rs							6
Absolute address	MOV.W	Rs,@aa:16	6 B	8 rs	abs						6
	MOV.W	Rs,@aa:24	6 B	A rs	0 0	abs					8

Notes

1. The destination operand <EAd> must be located at an even address.
2. In machine language, MOV.W Rs, @–R7 is identical to PUSH.W Rs.
3. Execution of MOV.W Rn, @–ERn first decrements ERn by 2, then transfers the resulting value.

2.2.35 (9) MOV (L)

MOV (MOVe data) **Move**

Operation

ERs → (EAd)

Assembly-Language Format

`MOV.L` ERs, <EAd>

Operand Size

Longword

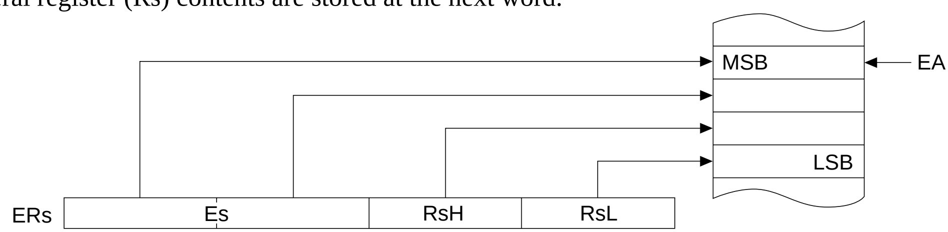
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers the contents of a 32-bit register ERs (source operand) to a destination location, tests the transferred data, and sets condition-code flags according to the result. The extended register (Es) contents are stored at the first word indicated by the effective address. The general register (Rs) contents are stored at the next word.



Available Registers

ERs: ER0 to ER7
ERd: ER0 to ER7

MOV (L)

MOV (MOVe data) Move

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format 1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte	No. of States
Register indirect	MOV.L	ERs,@ERd	0 1	0 0	6 9	1 erd 0 ers							8
Register indirect with displacement	MOV.L	ERs,@(d:16,ERd)	0 1	0 0	6 F	1 erd 0 ers	disp						10
	MOV.L	ERs,@(d:24,ERd)	0 1	0 0	7 8	1 erd 0	6 B	A 0 ers	0 0	disp			14
Register indirect with pre-decrement	MOV.L	ERs,@–ERd	0 1	0 0	6 D	1 erd 0 ers							10
Absolute address	MOV.L	ERs,@aa:16	0 1	0 0	6 B	8 0 ers	abs						10
	MOV.L	ERs,@aa:24	0 1	0 0	6 B	A 0 ers	0 0	abs					12

Notes

1. The destination operand <EAd> must be located at an even address.
2. In machine language, MOV.L ERs, @–ER7 is identical to PUSH.L ERs.
3. Execution of MOV.L ERn, @–ERn first decrements ERn by 4, then transfers the resulting value.

2.2.36 MOVFPE

MOVFPE (MOVe From Peripheral with E clock) **Move Data with E Clock**

Operation

(EAs) → Rd
Synchronized with E clock

Assembly-Language Format

`MOVFPE` @aa:16, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers memory contents specified by a 16-bit absolute address to a general register Rd in synchronization with an E clock, tests the transferred data, and sets condition-code flags according to the result.

Note: Avoid using this instruction in microcontrollers not having an E clock output pin, or in single-chip mode.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Absolute address	MOVFPE	@aa:16, Rd	6	A	4	rd	abs		*

Notes

1. This instruction cannot be used with addressing modes other than the above, and cannot transfer word data or longword data.
2. Data transfer by this instruction requires 9 to 16 states, so the execution time is variable. For details, refer to the relevant microcontroller hardware manual.

2.2.37 MOVTPE

MOVTPE (MOVe To Peripheral with E clock) **Move Data with E Clock**

Operation

Rs → (EAd)
Synchronized with E clock

Assembly-Language Format

`MOVTPE` Rs, @aa:16

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction transfers the contents of a general register Rs (source operand) to a destination location specified by a 16-bit absolute address in synchronization with an E clock, tests the transferred data, and sets condition-code flags according to the result.

Note: Avoid using this instruction in microcontrollers not having an E clock output pin, or in single-chip mode.

Available Registers

Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Absolute address	MOVTPE	Rs, @aa:16	6	A	C	rs	abs		*

Notes

1. This instruction cannot be used with addressing modes other than the above, and cannot transfer word data or longword data.
2. Data transfer by this instruction requires 9 to 16 states, so the execution time is variable. For details, refer to the relevant microcontroller hardware manual.

2.2.38 (1) MULXS (B)

MULXS (MULtiply eXtend as Signed) **Multiply Signed**

Operation

Rd × Rs → Rd

Assembly-Language Format

`MULXS.B` Rs, Rd

Operand Size

Byte

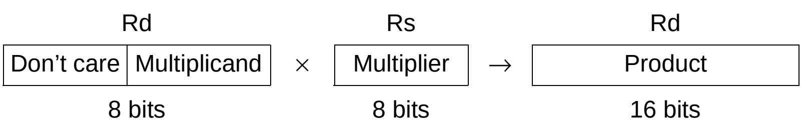
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	—	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction multiplies the lower 8 bits of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) as signed data and stores the result in the 16-bit register Rd. If Rd is a general register, Rs can be the upper part (RdH) or lower part (RdL) of Rd. The operation performed is 8-bit × 8-bit → 16-bit signed multiplication.



Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	MULXS.B	Rs, Rd	0	1	C	0	5	0	rs	rd	16

Notes

2.2.38 (2) MULXS (W)

MULXS (MULtiply eXtend as Signed) **Multiply Signed**

Operation

ERd × Rs → ERd

Assembly-Language Format

`MULXS.W` Rs, ERd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	—	—

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

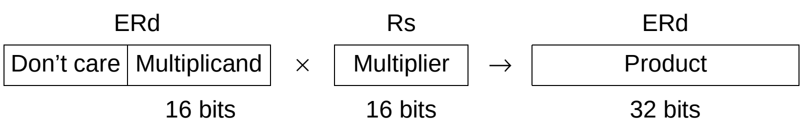
Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Previous value remains unchanged.

C: Previous value remains unchanged.

Description

This instruction multiplies the lower 16 bits of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) as signed data and stores the result in the 32-bit register ERd. Rs can be the upper part (Ed) or lower part (Rd) of ERd. The operation performed is 16-bit × 16-bit → 32-bit signed multiplication.



Available Registers

ERd: ER0 to ER7

Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	MULXS.W	Rs, ERd	0	1	C	0	5	2	rs	0 erd	24

Notes

2.2.39 (1) MULXU (B)

MULXU (MULtiply eXtend as Unsigned) **Multiply**

Operation

Rd × Rs → Rd

Assembly-Language Format

`MULXU.B` Rs, Rd

Operand Size

Byte

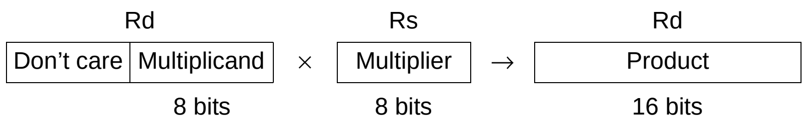
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction multiplies the lower 8 bits of a 16-bit register Rd (destination operand) by the contents of an 8-bit register Rs (source operand) and stores the result in the 16-bit register Rd. If Rd is a general register, Rs can be the upper part (RdH) or lower part (RdL) of Rd. The operation performed is 8-bit × 8-bit → 16-bit multiplication.



Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	MULXU.B	Rs, Rd	5	0	rs	rd			14

Notes

2.2.39 (2) MULXU (W)

MULXU (MULtiply eXtend as Unsigned) **Multiply**

Operation

ERd × Rs → ERd

Assembly-Language Format

`MULXU.W` Rs, ERd

Operand Size

Word

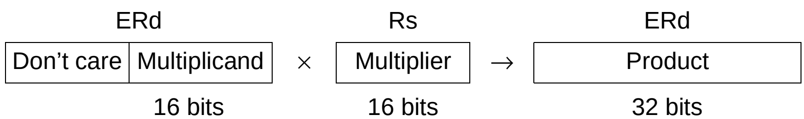
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction multiplies the lower 16 bits of a 32-bit register ERd (destination operand) by the contents of a 16-bit register Rs (source operand) and stores the result in the 32-bit register ERd. Rs can be the upper part (Ed) or lower part (Rd) of ERd. The operation performed is 16-bit × 16-bit → 32-bit multiplication.



Available Registers

ERd: ER0 to ER7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	MULXU.W	Rs, ERd	5 2	rs 0 erd			22

Notes

2.2.40 (1) NEG (B)

NEG (NEGate) **Negate Binary Signed**

Operation

0 – Rd → Rd

Assembly-Language Format

`NEG.B` Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

Description

This instruction takes the two's complement of the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd (subtracting the register contents from H'00). If the original contents of Rd was H'80, however, the result remains H'80.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	NEG.B	Rd	1	7	8	rd					2

Notes

An overflow occurs if the previous contents of Rd was H'80.

2.2.40 (2) NEG (W)

NEG (NEGate) **Negate Binary Signed**

Operation

0 – Rd → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.

Assembly-Language Format

`NEG.W` Rd

Operand Size

Word

Description

This instruction takes the two's complement of the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd (subtracting the register contents from H'0000). If the original contents of Rd was H'8000, however, the result remains H'8000.

Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	NEG.W	Rd	1	7	9	rd			2

Notes

An overflow occurs if the previous contents of Rd was H'8000.

2.2.40 (3) NEG (L)

NEG (NEGate) **Negate Binary Signed**

Operation

0 – ERd → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 27; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.

Assembly-Language Format

`NEG.L` ERd

Operand Size

Longword

Description

This instruction takes the two's complement of the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd (subtracting the register contents from H'00000000). If the original contents of ERd was H'80000000, however, the result remains H'80000000.

Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NEG.L	ERd	1 7	B 0 erd			2

Notes

An overflow occurs if the previous contents of ERd was H'80000000.

RENESAS

2.2.41 NOP

NOP (No OPeration) **No Operation**

Operation

PC + 2 → PC

Assembly-Language Format

NOP

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction only increments the program counter, causing the next instruction to be executed. The internal state of the CPU does not change.

Available Registers

—

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	NOP		0 0	0 0			2

Notes

2.2.42 (1) NOT (B)

NOT (NOT = logical complement) **Logical Complement**

Operation

¬ Rd → Rd

Assembly-Language Format

`NOT.B` Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction takes the one's complement of the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	NOT.B	Rd	1	7	0	rd			2

Notes

2.2.42 (2) NOT (W)

NOT (NOT = logical complement) **Logical Complement**

Operation

$\neg$ Rd $\rightarrow$ Rd

Assembly-Language Format

NOT.W Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero (the previous Rd value was H'FFFF); otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction takes the one's complement of the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NOT.W	Rd	1 ⁞ 7	1 ⁞ rd			2

Notes

2.2.42 (3) NOT (L)

NOT (NOT = logical complement) **Logical Complement**

Operation

¬ ERd → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

I: Previous value remains unchanged.
H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Assembly-Language Format

`NOT.L` ERd

Operand Size

Longword

Description

This instruction takes the one's complement of the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	NOT.L	ERd	1 ⋮ 7	3 ⋮ 0 ⋮ erd			2

Notes

2.2.43 (1) OR (B)

OR (inclusive OR logical) **Logical OR**

Operation

Rd ∨ (EAs) → Rd

Assembly-Language Format

`OR.B` <EAs>, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction ORs the source operand with the contents of an 8-bit register Rd (destination register) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	OR.B	#xx:8, Rd	C	rd	IMM				2
Register direct	OR.B	Rs, Rd	1	4	rs	rd			2

Notes

2.2.43 (2) OR (W)

OR (inclusive OR logical) **Logical OR**

Operation

Rd ∨ (EAs) → Rd

Assembly-Language Format

`OR.W` <EAs>, Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction ORs the source operand with the contents of a 16-bit register Rd (destination register) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	OR.W	#xx:16, Rd	7	9	4	rd	IMM		4
Register direct	OR.W	Rs, Rd	6	4	rs	rd			2

Notes

2.2.43 (3) OR (L)

OR (inclusive OR logical) **Logical OR**

Operation

ERd $\vee$ (EAs) $\rightarrow$ ERd

Assembly-Language Format

`OR.L` <EAs>, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction ORs the source operand with the contents of a 32-bit register ERd (destination register) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	
Immediate	OR.L	#xx:32,ERd	7 A	4 0 erd	IMM				6
Register direct	OR.L	ERs, ERd	0 1	F 0	6 4	0 ers 0 erd			4

Notes

2.2.44 ORC

ORC (inclusive OR Control register) **Logical OR with CCR**

Operation

CCR ∨ #IMM → CCR

Assembly-Language Format

ORC #xx:8, CCR

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

I: Stores the corresponding bit of the result.
UI: Stores the corresponding bit of the result.
H: Stores the corresponding bit of the result.
U: Stores the corresponding bit of the result.
N: Stores the corresponding bit of the result.
Z: Stores the corresponding bit of the result.
V: Stores the corresponding bit of the result.
C: Stores the corresponding bit of the result.

Description

This instruction ORs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format					No. of States
			1st byte		2nd byte	3rd byte	4th byte	
Immediate	ORC	#xx:8, CCR	0	4	IMM			2

Notes

2.2.45 (1) POP (W)

POP (POP data) **Pop Data from Stack**

Operation

@SP+ → Rn

Assembly-Language Format

POP.W Rn

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction restores data from the stack to a 16-bit general register Rn, tests the restored data, and sets condition-code flags according to the result.

Available Registers

Rn: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
—	POP.W	Rn	6	D	7	rn			6

Notes

POP.W Rn is identical to MOV.W @SP+, Rn.

2.2.45 (2) POP (L)

POP (POP data) **Pop Data from Stack**

Operation

@SP+ → ERn

Assembly-Language Format

`POP.L` ERn

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction restores data from the stack to a 32-bit general register ERn, tests the restored data, and sets condition-code flags according to the result.

Available Registers

ERn: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format									No. of States
			1st byte		2nd byte		3rd byte		4th byte			
—	POP.L	ERn	0	1	0	0	6	D	7	0	ern	10

Notes

POP.L ERn is identical to MOV.L @SP+, ERn.

RENESAS

2.2.46 (1) PUSH (W)

PUSH (PUSH data) **Push Data on Stack**

Operation

Rn → @–SP

Assembly-Language Format

`PUSH.W` Rn

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction saves data from a 16-bit register Rn onto the stack, tests the saved data, and sets condition-code flags according to the result.

Available Registers

Rn: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
—	PUSH.W	Rn	6	D	F	rn					6

Notes

1. PUSH.W Rn is identical to MOV.W Rn, @–SP.
2. When PUSH.W R7 or PUSH.W E7 is executed, the value saved on the stack is the lower part (R7) or upper part (E7) of the value of ER7 before execution minus two.

2.2.46 (2) PUSH (L)

PUSH (PUSH data) **Push Data on Stack**

Operation

ERn → @–SP

Assembly-Language Format

`PUSH.L` ERn

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the data value is negative; otherwise cleared to 0.
Z: Set to 1 if the data value is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction pushes data from a 32-bit register ERn onto the stack, tests the saved data, and sets condition-code flags according to the result.

Available Registers

ERn: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
—	PUSH.L	ERn	0	1	0	0	6	D	F	0 ern	10

Notes

1. PUSH.L ERn is identical to MOV.L ERn, @–SP.
2. When PUSH.L ER7 is executed, the value saved on the stack is the value of ER7 before execution minus four.

2.2.47 (1) ROTL (B)

ROTL (ROTate Left) **Rotate**

Operation

Rd (left rotation) → Rd

Assembly-Language Format

`ROTL.B` Rd

Operand Size

Byte

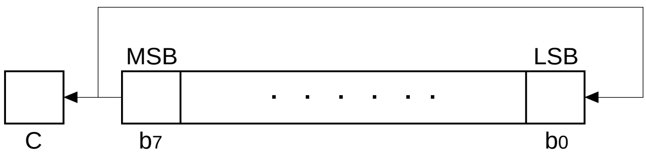
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 7.

Description

This instruction rotates the bits in an 8-bit register Rd (destination register) one bit to the left. The most significant bit is rotated to the least significant bit (bit 0), and also copied to the carry flag.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	ROTL.B	Rd	1	2	8	rd			2

Notes

2.2.47 (2) ROTL (W)

ROTL (ROTate Left) **Rotate**

Operation

Rd (left rotation) → Rd

Assembly-Language Format

ROTL.W Rd

Operand Size

Word

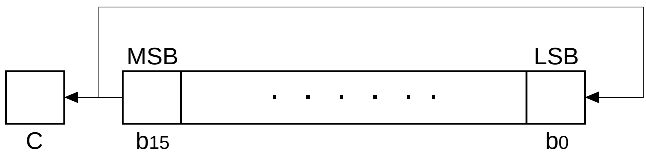
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 15.

Description

This instruction rotates the bits in a 16-bit register Rd (destination register) one bit to the left. The most significant bit is rotated to the least significant bit (bit 0), and also copied to the carry flag.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	ROTL.W	Rd	1	2	9	rd			2

Notes

2.2.47 (3) ROTL (L)

ROTL (ROTate Left) **Rotate**

Operation

ERd (left rotation) → ERd

Assembly-Language Format

`ROTL.L` ERd

Operand Size

Longword

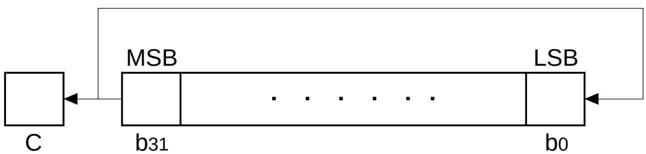
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 31.

Description

This instruction rotates the bits in a 32-bit register ERd (destination register) one bit to the left. The most significant bit is rotated to the least significant bit (bit 0), and also copied to the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTL.L	ERd	1 ¦ 2	B ¦ 0 ¦ erd			2

Notes

2.2.48 (1) ROTR (B)

ROTR (ROTate Right) **Rotate**

Operation

Rd (right rotation) → Rd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Assembly-Language Format

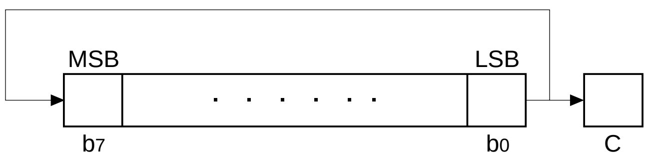
`ROTR.B` Rd

Operand Size

Byte

Description

This instruction rotates the bits in an 8-bit register Rd (destination register) one bit to the right. The least significant bit is rotated to the most significant bit (bit 7), and also copied to the carry flag.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	ROTR.B	Rd	1	3	8	rd			2

Notes

2.2.48 (2) ROTR (W)

ROTR (ROTate Right) **Rotate**

Operation

Rd (right rotation) → Rd

Assembly-Language Format

`ROTR.W` Rd

Operand Size

Word

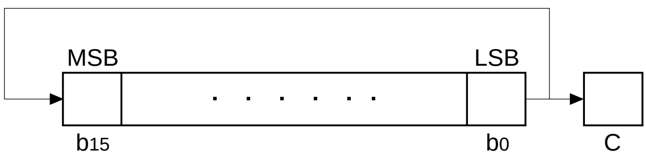
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction rotates the bits in a 16-bit register Rd (destination register) one bit to the right. The least significant bit is rotated to the most significant bit (bit 15), and also copied to the carry flag.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	ROTR.W	Rd	1	3	9	rd			2

Notes

2.2.48 (3) ROTR (L)

ROTR (ROTate Right) **Rotate**

Operation

ERd (right rotation) → ERd

Assembly-Language Format

`ROTR.L` ERd

Operand Size

Longword

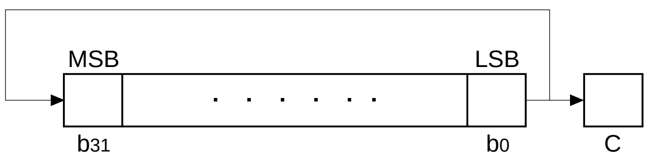
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction rotates the bits in a 32-bit register ERd (destination register) one bit to the right. The least significant bit is rotated to the most significant bit (bit 31), and also copied to the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTR.L	ERd	1 3	B 0 erd			2

Notes

2.2.49 (1) ROTXL (B)

ROTXL (ROTate with eXtend carry Left) **Rotate through Carry**

Operation

Rd (left rotation through carry bit) → Rd

Assembly-Language Format

`ROTXL.B` Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.

N: Set to 1 if the result is negative; otherwise cleared to 0.

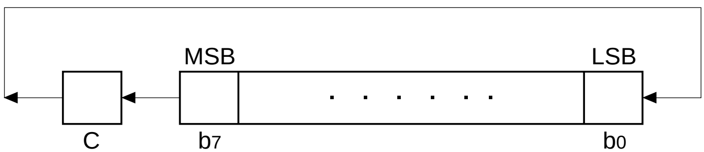
Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Always cleared to 0.

C: Receives the previous value in bit 7.

Description

This instruction rotates the bits in an 8-bit register Rd (destination register) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit rotates into the carry flag.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.B	Rd	1 ⋮ 2	0 ⋮ rd			2

Notes

2.2.49 (2) ROTXL (W)

ROTXL (ROTate with eXtend carry Left) **Rotate through Carry**

Operation

Rd (left rotation through carry bit) → Rd

Assembly-Language Format

`ROTXL.W` Rd

Operand Size

Word

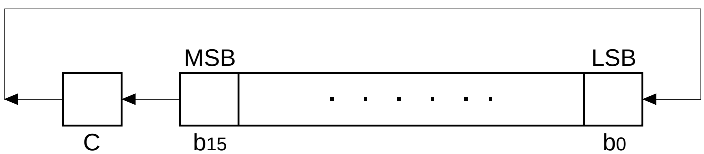
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 15.

Description

This instruction rotates the bits in a 16-bit register Rd (destination register) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit rotates into the carry flag.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXL.W	Rd	1 ⋮ 2	1 ⋮ rd			2

Notes

2.2.49 (3) ROTXL (L)

ROTXL (ROTate with eXtend carry Left) **Rotate through Carry**

Operation

ERd (left rotation through carry bit) → ERd

Assembly-Language Format

`ROTXL.L` ERd

Operand Size

Longword

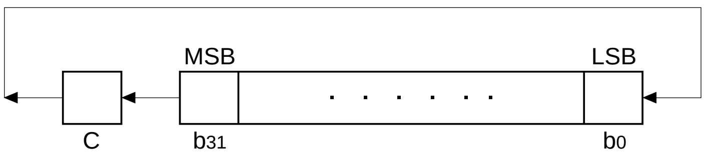
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 31.

Description

This instruction rotates the bits in a 32-bit register ERd (destination register) one bit to the left through the carry flag. The carry flag is rotated into the least significant bit (bit 0). The most significant bit rotates into the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	ROTXL.L	ERd	1	2	3	0 erd			2

Notes

2.2.50 (1) ROTXR (B)

ROTXR (ROTate with eXtend carry Right) **Rotate through Carry**

Operation

Rd (right rotation through carry bit) → Rd

Assembly-Language Format

`ROTXR.B Rd`

Operand Size

Byte

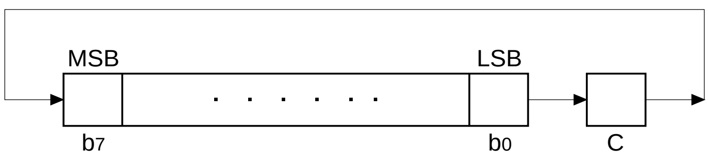
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction rotates the bits in an 8-bit register Rd (destination register) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 7). The least significant bit rotates into the carry flag.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	ROTXR.B	Rd	1	3	0	rd			2

Notes

2.2.50 (2) ROTXR (W)

ROTXR (ROTate with eXtend carry Right) **Rotate through Carry**

Operation

Rd (right rotation through carry bit) → Rd

Assembly-Language Format

`ROTXR.W Rd`

Operand Size

Word

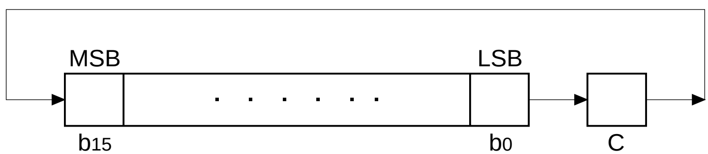
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction rotates the bits in a 16-bit register Rd (destination register) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 15). The least significant bit rotates into the carry flag.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	ROTXR.W	Rd	1	3	1	rd			2

Notes

2.2.50 (3) ROTXR (L)

ROTXR (ROTate with eXtend carry Right) **Rotate through Carry**

Operation

ERd (right rotation through carry bit) → ERd

Assembly-Language Format

ROTXR.L ERd

Operand Size

Longword

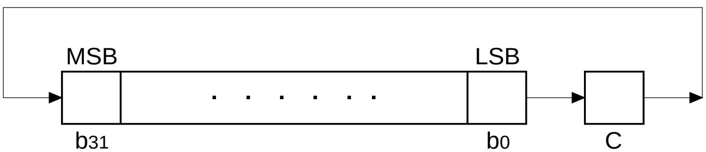
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction rotates the bits in a 32-bit register ERd (destination register) one bit to the right through the carry flag. The carry flag is rotated into the most significant bit (bit 31). The least significant bit rotates into the carry flag.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	ROTXR.L	ERd	1 ⁝ 3	3 ⁝ 0 ⁝ erd			2

Notes

2.2.51 RTE

RTE (ReTurn from Exception) **Return from Exception Handling**

Operation

@SP+ → CCR
@SP+ → PC

Assembly-Language Format

RTE

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

I: Restored from the corresponding bit on the stack.
UI: Restored from the corresponding bit on the stack.
H: Restored from the corresponding bit on the stack.
U: Restored from the corresponding bit on the stack.
N: Restored from the corresponding bit on the stack.
Z: Restored from the corresponding bit on the stack.
V: Restored from the corresponding bit on the stack.
C: Restored from the corresponding bit on the stack.

Description

This instruction returns from an exception-handling routine by restoring the condition-code register (CCR) and program counter (PC) from the stack. Program execution continues from the address restored to the program counter. The CCR and PC contents at the time of execution of this instruction are lost.

Operand Format and Number of States Required for Execution

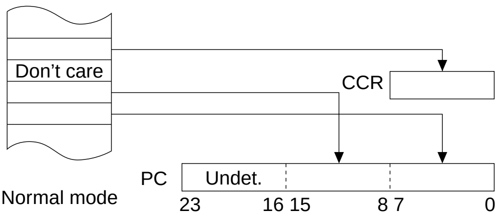
Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
—	RTE		5 6	7 0			10

RTE

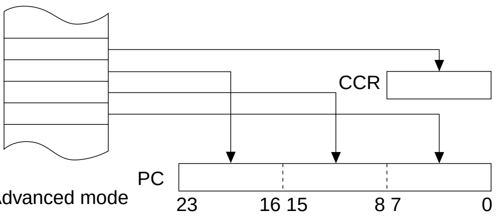
RTE (ReTurn from Exception) Return from Exception Handling

Notes

The stack structure differs between normal mode and advanced mode.



Normal mode



Advanced mode

2.2.52 RTS

RTS (ReTurn from Subroutine) **Return from Subroutine**

Operation

@SP+ → PC

Assembly-Language Format

RTS

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction returns from a subroutine by restoring the program counter (PC) from the stack. Program execution continues from the address restored to the program counter. The PC contents at the time of execution of this instruction are lost.

Available Registers

—

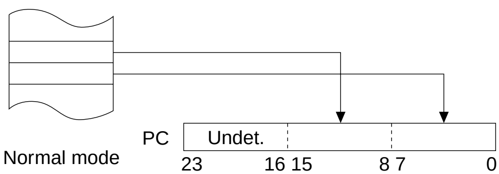
Operand Format and Number of States Required for Execution

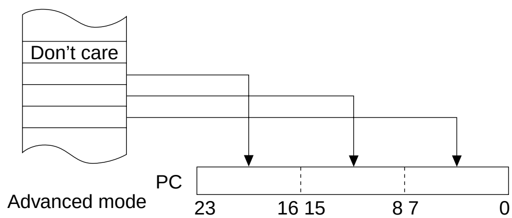
Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States	
			1st byte	2nd byte	3rd byte	4th byte	Normal	Advanced
—	RTS		5 4	7 0			8	10

Notes

The stack structure and number of states required for execution differ between normal mode and advanced mode.

In normal mode, only the lower 16 bits of the program counter are restored.





2.2.53 (1) SHAL (B)

SHAL (SHift Arithmetic Left) **Shift Arithmetic**

Operation

Rd (left arithmetic shift) → Rd

Assembly-Language Format

`SHAL.B` Rd

Operand Size

Byte

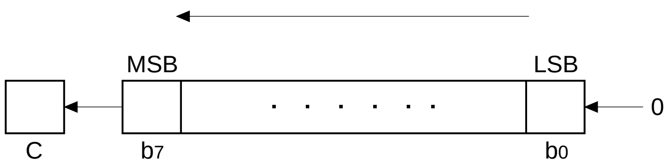
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Receives the previous value in bit 7.

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.B	Rd	1 ¦ 0	8 ¦ rd			2

Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

2.2.53 (2) SHAL (W)

SHAL (SHift Arithmetic Left) **Shift Arithmetic**

Operation

Rd (left arithmetic shift) → Rd

Assembly-Language Format

`SHAL.W` Rd

Operand Size

Word

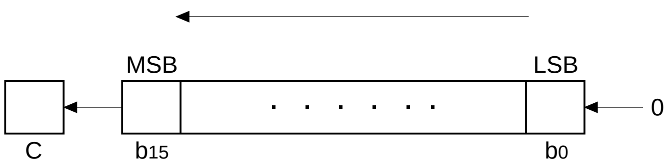
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Receives the previous value in bit 15.

Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.W	Rd	1 0	9 rd			2

Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

2.2.53 (3) SHAL (L)

SHAL (SHift Arithmetic Left) **Shift Arithmetic**

Operation

ERd (left arithmetic shift) → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	↕	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Receives the previous value in bit 31.

Assembly-Language Format

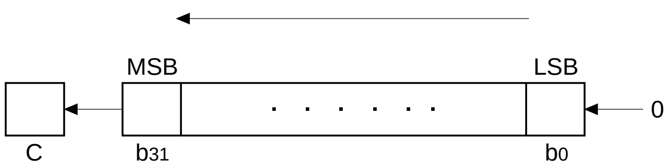
`SHAL.L` ERd

Operand Size

Longword

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAL.L	ERd	1 0	B 0 erd			2

Notes

The SHAL instruction differs from the SHLL instruction in its effect on the overflow flag.

2.2.54 (1) SHAR (B)

SHAR (SHift Arithmetic Right) **Shift Arithmetic**

Operation

Rd (right arithmetic shift) → Rd

Assembly-Language Format

`SHAR.B` Rd

Operand Size

Byte

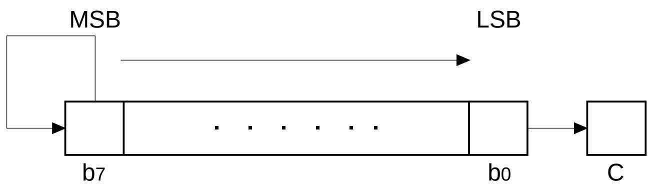
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 7 shifts into itself. Since bit 7 remains unaltered, the sign does not change.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
Register direct	SHAR.B	Rd	1	1	8	rd					2

Notes

2.2.54 (2) SHAR (W)

SHAR (SHift Arithmetic Right) **Shift Arithmetic**

Operation

Rd (right arithmetic shift) → Rd

Assembly-Language Format

`SHAR.W` Rd

Operand Size

Word

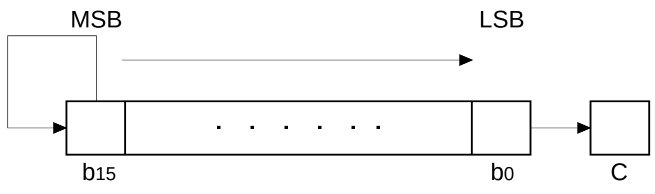
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 15 shifts into itself. Since bit 15 remains unaltered, the sign does not change.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	SHAR.W	Rd	1	1	9	rd			2

Notes

2.2.54 (3) SHAR (L)

SHAR (SHift Arithmetic Right) **Shift Arithmetic**

Operation

ERd (right arithmetic shift) → ERd

Assembly-Language Format

`SHAR.L` ERd

Operand Size

Longword

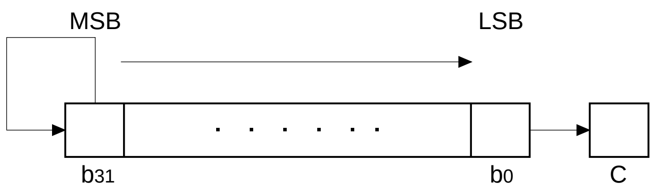
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the right. Bit 0 shifts into the carry flag. Bit 31 shifts into itself. Since bit 31 remains unaltered, the sign does not change.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHAR.L	ERd	1 1	B 0 erd			2

Notes

2.2.55 (1) SHLL (B)

SHLL (SHift Logical Left) **Shift Logical**

Operation

Rd (left logical shift) → Rd

Assembly-Language Format

`SHLL.B` Rd

Operand Size

Byte

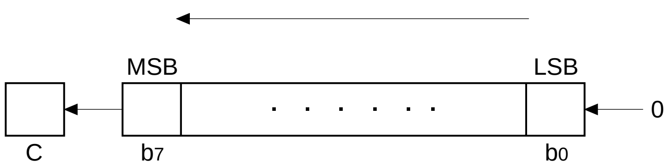
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 7.

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the left. The most significant bit shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.B	Rd	1 ⋮ 0	0 ⋮ rd			2

Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

2.2.55 (2) SHLL (W)

SHLL (SHift Logical Left) **Shift Logical**

Operation

Rd (left logical shift) → Rd

Assembly-Language Format

`SHLL.W` Rd

Operand Size

Word

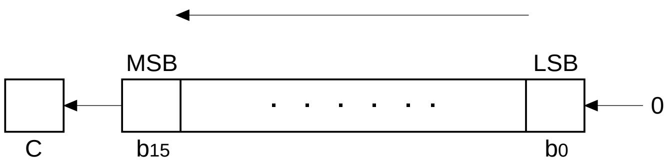
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 15.

Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the left. The most significant bit shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLL.W	Rd	1 0	1 rd			2

Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

2.2.55 (3) SHLL (L)

SHLL (SHift Logical Left) **Shift Logical**

Operation

ERd (left logical shift) → ERd

Assembly-Language Format

`SHLL.L` ERd

Operand Size

Longword

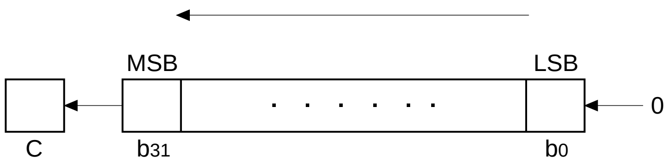
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 31.

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the left. The most significant bit shifts into the carry flag. The least significant bit (bit 0) is cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format 1st byte		2nd byte		3rd byte	4th byte	No. of States
Register direct	SHLL.L	ERd	1	0	3	0 erd			2

Notes

The SHLL instruction differs from the SHAL instruction in its effect on the overflow flag.

2.2.56 (1) SHLR (B)

SHLR (SHift Logical Right) **Shift Logical**

Operation

Rd (right logical shift) → Rd

Assembly-Language Format

`SHLR.B` Rd

Operand Size

Byte

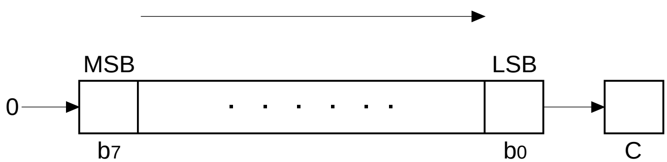
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	↕

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction shifts the bits in an 8-bit register Rd (destination operand) one bit to the right. The least significant bit shifts into the carry flag. The most significant bit (bit 7) is cleared to 0.



Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.B	Rd	1 ¦ 1	0 ¦ rd			2

Notes

2.2.56 (2) SHLR (W)

SHLR (SHift Logical Right) **Shift Logical**

Operation

Rd (right logical shift) → Rd

Assembly-Language Format

`SHLR.W` Rd

Operand Size

Word

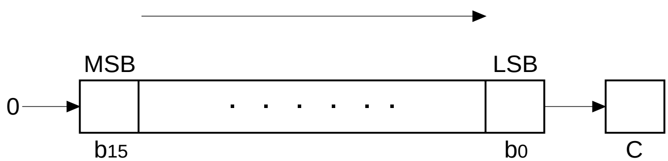
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	↕

H: Previous value remains unchanged.
N: Always cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction shifts the bits in a 16-bit register Rd (destination operand) one bit to the right. The least significant bit shifts into the carry flag. The most significant bit (bit 15) is cleared to 0.



Available Registers

Rd: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.W	Rd	1 1	1 rd			2

Notes

2.2.56 (3) SHLR (L)

SHLR (SHift Logical Right) **Shift Logical**

Operation

ERd (right logical shift) → ERd

Assembly-Language Format

`SHLR.L` ERd

Operand Size

Longword

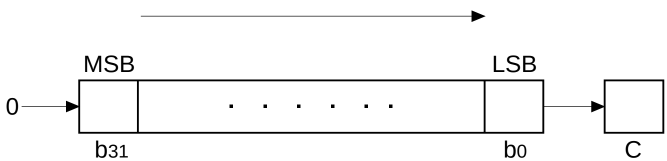
Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	0	↕	0	↕

H: Previous value remains unchanged.
N: Always cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Receives the previous value in bit 0.

Description

This instruction shifts the bits in a 32-bit register ERd (destination operand) one bit to the right. The least significant bit shifts into the carry flag. The most significant bit (bit 31) is cleared to 0.



Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SHLR.L	ERd	1 1	3 0 erd			2

Notes

2.2.57 SLEEP

SLEEP (SLEEP) **Power-Down Mode**

Operation

Program execution state → power-down mode

Assembly-Language Format

SLEEP

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

When the SLEEP instruction is executed, the CPU enters a power-down state. Its internal state remains unchanged, but the CPU stops executing instructions and waits for an exception-handling request. When it receives an exception-handling request, the CPU exits the power-down state and begins the exception-handling sequence. Interrupt requests other than NMI cannot end the power-down state if they are masked in the CPU.

Available Registers

—

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format								No. of States
			1st byte		2nd byte		3rd byte		4th byte		
—	SLEEP		0	1	8	0					2

Notes

For information about the power-down state, see the relevant microcontroller hardware manual.

2.2.58 (1) STC (B)

STC (STore from Control register) **Store CCR**

Operation

CCR → Rd

Assembly-Language Format

STC.B CCR, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction copies the CCR contents to an 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Register direct	STC.B	CCR, Rd	0	2	0	rd			2

Notes

2.2.58 (2) STC (W)

STC (STore from Control register) **Store CCR**

Operation

CCR → (EAd)

Assembly-Language Format

STC.W CCR, <EAd>

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction copies the CCR contents to a destination location. Although CCR is a byte register, the destination operand is a word operand. The CCR contents are stored at the even address.

Available Registers

ERd: ER0 to ER7

STC (W)

STC (STore from Control register) — Store CCR

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format										No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte	
Register indirect	STC.W	CCR,@ERd	0 1	4 0	6 9	1 erd 0							6
Register indirect with displacement	STC.W	CCR,@(d:16,ERd)	0 1	4 0	6 F	1 erd 0	disp						8
	STC.W	CCR,@(d:24,ERd)	0 1	4 0	7 8	0 erd 0	6 B	A 0	0 0	disp			12
Register indirect with pre-decrement	STC.W	CCR,@–ERd	0 1	4 0	6 D	1 erd 0							8
Absolute address	STC.W	CCR,@aa:16	0 1	4 0	6 B	8 0	abs						8
	STC.W	CCR,@aa:24	0 1	4 0	6 B	A 0	0 0	abs					10

Notes

2.2.59 (1) SUB (B)

SUB (SUBtract binary) **Subtract Binary**

Operation

Rd – Rs → Rd

Assembly-Language Format

`SUB.B` Rs, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 3; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 7; otherwise cleared to 0.

Description

This instruction subtracts the contents of an 8-bit register Rs (source operand) from the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	SUB.B	Rs, Rd	1 ⋮ 8	rs ⋮ rd			2

SUB (B)

SUB (SUBtract binary) **Subtract Binary**

Notes

The SUB.B instruction can operate only on general registers. Immediate data can be subtracted from general register contents by using the SUBX instruction. Before executing SUBX #xx:8, Rd, first set the Z flag to 1 and clear the C flag to 0. The following coding examples can also be used to subtract nonzero immediate data #IMM.

(1)
```
ORC  #H'05, CCR
     SUBX #(IMM–1), Rd
```

(2)
```
ADD  #(0–IMM), Rd
     XORC #H'01, CCR
```

2.2.59 (2) SUB (W)

SUB (SUBtract binary) **Subtract Binary**

Operation

Rd – (EAs) → Rd

Assembly-Language Format

`SUB.W` <EAs>, Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 11; otherwise cleared to 0.

N: Set to 1 if the result is negative; otherwise cleared to 0.

Z: Set to 1 if the result is zero; otherwise cleared to 0.

V: Set to 1 if an overflow occurs; otherwise cleared to 0.

C: Set to 1 if there is a borrow at bit 15; otherwise cleared to 0.

Description

This instruction subtracts a source operand from the contents of a 16-bit register Rd (destination operand) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7

Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	SUB.W	#xx:16, Rd	7	9	3	rd	IMM		4
Register direct	SUB.W	Rs, Rd	1	9	rs	rd			2

Notes

2.2.59 (3) SUB (L)

SUB (SUBtract binary) **Subtract Binary**

Operation

ERd – <EAs> → ERd

Assembly-Language Format

`SUB.L` <EAs>, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow at bit 27; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow at bit 31; otherwise cleared to 0.

Description

This instruction subtracts a source operand from the contents of a 32-bit register ERd (destination operand) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	
Immediate	SUB.L	#xx:32, ERd	7 ¦ A	3 ¦ 0 ¦ erd	IMM				6
Register direct	SUB.L	ERs, ERd	1 ¦ A	1 ¦ ers ¦ 0 ¦ erd					2

Notes

2.2.60 SUBS

SUBS (SUBtract with Sign extension) **Subtract Binary Address Data**

Operation

ERd – 1 → ERd
ERd – 2 → ERd
ERd – 4 → ERd

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	—	—	—	—

H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Assembly-Language Format

SUBS #1, ERd
SUBS #2, ERd
SUBS #4, ERd

Operand Size

Longword

Description

This instruction subtracts the immediate value 1, 2, or 4 from the contents of a 32-bit register ERd (destination register). Differing from the SUB instruction, it does not affect the condition-code flags.

Available Registers

ERd: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format							No. of States
			1st byte		2nd byte			3rd byte	4th byte	
Register direct	SUBS	#1, ERd	1	B	0	0	erd			2
Register direct	SUBS	#2, ERd	1	B	8	0	erd			2
Register direct	SUBS	#4, ERd	1	B	9	0	erd			2

Notes

2.2.61 SUBX

SUBX (SUBtract with eXtend carry) **Subtract with Borrow**

Operation

Rd – (EAs) – C → Rd

Assembly-Language Format

SUBX <EAs>, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

H: Set to 1 if there is a borrow from bit 3; otherwise cleared to 0.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Set to 1 if an overflow occurs; otherwise cleared to 0.
C: Set to 1 if there is a borrow from bit 7; otherwise cleared to 0.

Description

This instruction subtracts the source operand and carry flag from the contents of an 8-bit register Rd (destination operand) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	SUBX	#xx:8, Rd	B	rd	IMM				2
Register direct	SUBX	Rs, Rd	1	E	rs	rd			2

Notes

2.2.62 TRAPA

TRAPA (TRAP Always) **Trap Unconditionally**

Operation

PC → @–SP
CCR → @–SP
<Vector> → PC

Assembly-Language Format

TRAPA #x:2

Operand Size

—

Condition Code

I	UI	H	U	N	Z	V	C
1	Δ*1	—	—	—	—	—	—

I: Always set to 1.
U: See notes.
H: Previous value remains unchanged.
N: Previous value remains unchanged.
Z: Previous value remains unchanged.
V: Previous value remains unchanged.
C: Previous value remains unchanged.

Description

This instruction pushes the program counter (PC) and condition-code register (CCR) on the stack, then sets the I bit to 1 and branches to a new address. The new address is the contents of the vector address corresponding to the specified vector number. The PC value pushed on the stack is the starting address of the next instruction after the TRAPA instruction.

#x	Vector Address	
	Normal Mode	Advanced Mode
0	H'0010 to H'0011	H'000020 to H'000023
1	H'0012 to H'0013	H'000024 to H'000027
2	H'0014 to H'0015	H'000028 to H'00002B
3	H'0016 to H'0017	H'00002C to H'00002F

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format				No. of States
			1st byte	2nd byte	3rd byte	4th byte	
Register direct	TRAPA	#x:2	5 ⁞ 7	00 ⁞ IMM ⁞ 0			14

Notes

1. CCR bit 6 is set to 1 when used as an interrupt mask bit, but retains its previous value when used as a user bit.
2. The stack and vector structure differ between normal mode and advanced mode.

2.2.63 (1) XOR (B)

XOR (eXclusive OR logical) **Exclusive Logical OR**

Operation

Rd ⊕ (EAs) → Rd

Assembly-Language Format

XOR.B <EAs>, Rd

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction exclusively ORs the source operand with the contents of an 8-bit register Rd (destination register) and stores the result in the 8-bit register Rd.

Available Registers

Rd: R0L to R7L, R0H to R7H
Rs: R0L to R7L, R0H to R7H

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	XOR.B	#xx:8, Rd	D	rd	IMM				2
Register direct	XOR.B	Rs, Rd	1	5	rs	rd			2

Notes

2.2.63 (2) XOR (W)

XOR (eXclusive OR logical) **Exclusive Logical OR**

Operation

Rd ⊕ (EAs) → Rd

Assembly-Language Format

`XOR.W` <EAs>, Rd

Operand Size

Word

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction exclusively ORs the source operand with the contents of a 16-bit register Rd (destination register) and stores the result in the 16-bit register Rd.

Available Registers

Rd: R0 to R7, E0 to E7
Rs: R0 to R7, E0 to E7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte		2nd byte		3rd byte	4th byte	
Immediate	XOR.W	#xx:16, Rd	7	9	5	rd	IMM		4
Register direct	XOR.W	Rs, Rd	6	5	rs	rd			2

Notes

2.2.63 (3) XOR (L)

XOR (eXclusive OR logical) **Exclusive Logical OR**

Operation

ERd ⊕ (EAs) → ERd

Assembly-Language Format

XOR.L <EAs>, ERd

Operand Size

Longword

Condition Code

I	UI	H	U	N	Z	V	C
—	—	—	—	↕	↕	0	—

H: Previous value remains unchanged.
N: Set to 1 if the result is negative; otherwise cleared to 0.
Z: Set to 1 if the result is zero; otherwise cleared to 0.
V: Always cleared to 0.
C: Previous value remains unchanged.

Description

This instruction exclusively ORs the source operand with the contents of a 32-bit register ERd (destination register) and stores the result in the 32-bit register ERd.

Available Registers

ERd: ER0 to ER7
ERs: ER0 to ER7

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format						No. of States
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	
Immediate	XOR.L	#xx:32, ERd	7 A	5 0 erd	IMM				6
Register direct	XOR.L	ERs, ERd	0 1	F 0	6 5	0 ers 0 erd			4

Notes

2.2.64 XORC

XORC (eXclusive OR Control register) **Exclusive Logical OR with CCR**

Operation

CCR ⊕ #IMM → CCR

Assembly-Language Format

XORC #xx:8, CCR

Operand Size

Byte

Condition Code

I	UI	H	U	N	Z	V	C
↕	↕	↕	↕	↕	↕	↕	↕

I: Stores the corresponding bit of the result.
UI: Stores the corresponding bit of the result.
H: Stores the corresponding bit of the result.
U: Stores the corresponding bit of the result.
N: Stores the corresponding bit of the result.
Z: Stores the corresponding bit of the result.
V: Stores the corresponding bit of the result.
C: Stores the corresponding bit of the result.

Description

This instruction exclusively ORs the contents of the condition-code register (CCR) with immediate data and stores the result in the condition-code register. No interrupt requests, including NMI, are accepted immediately after execution of this instruction.

Operand Format and Number of States Required for Execution

Addressing Mode	Mnemonic	Operands	Instruction Format					No. of States
			1st byte		2nd byte	3rd byte	4th byte	
Immediate	XORC	#xx:8, CCR	0	5	IMM			2

Notes

2.3 Instruction Set Summary

Table 2.1 Instruction Set Summary

Function	Instruction	Addressing Modes												
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:24,ERn)	@ERn+/@–ERn	@aa:8	@aa:16	@aa:24	@(d:8,PC)	@(d:16,PC)	@@aa:8	—
Data transfer	MOV	BWL	BWL	BWL	BWL	BWL	BWL	B	BWL	BWL	—	—	—	—
	POP, PUSH	—	—	—	—	—	—	—	—	—	—	—	—	WL
	MOVFPE, MOVTPE	—	—	—	—	—	—	—	B	—	—	—	—	—
Arithmetic operations	ADD, CMP	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—
	SUB	WL	BWL	—	—	—	—	—	—	—	—	—	—	—
	ADDX, SUBX	B	B	—	—	—	—	—	—	—	—	—	—	—
	ADDS, SUBS	—	L	—	—	—	—	—	—	—	—	—	—	—
	INC, DEC	—	BWL	—	—	—	—	—	—	—	—	—	—	—
	DAA, DAS	—	B	—	—	—	—	—	—	—	—	—	—	—
	MULXU, DIVXU, MULXS, DIVXS	—	BW	—	—	—	—	—	—	—	—	—	—	—
	NEG	—	BWL	—	—	—	—	—	—	—	—	—	—	—
	EXTU, EXTS	—	WL	—	—	—	—	—	—	—	—	—	—	—
Logic operations	AND, OR, XOR	BWL	BWL	—	—	—	—	—	—	—	—	—	—	—
	NOT	—	BWL	—	—	—	—	—	—	—	—	—	—	—
Shift operations		—	BWL	—	—	—	—	—	—	—	—	—	—	—
Bit manipulation		—	B	B	—	—	—	B	—	—	—	—	—	—

Function	Instruction	Addressing Modes												
		#xx	Rn	@ERn	@(d:16,ERn)	@(d:24,ERn)	@ERn+/@–ERn	@aa:8	@aa:16	@aa:24	@(d:8,PC)	@(d:16,PC)	@@aa:8	—
Branch	Bcc, BSR	—	—	—	—	—	—	—	—	—	○	○	—	—
	JMP, JSR	—	—	○	—	—	—	—	—	○	—	—	○	—
	RTS	—	—	—	—	—	—	—	—	—	—	—	—	○
System control	TRAPA, RTE, SLEEP	—	—	—	—	—	—	—	—	—	—	—	—	○
	LDC	B	B	W	W	W	W	—	W	W	—	—	—	—
	STC	—	B	W	W	W	W	—	W	W	—	—	—	—
	ANDC, ORC, XORC	B	—	—	—	—	—	—	—	—	—	—	—	—
	NOP	—	—	—	—	—	—	—	—	—	—	—	—	○
Block data transfer		—	—	—	—	—	—	—	—	—	—	—	—	B

Legend:

B: Byte

W: Word

L: Longword

Table 2.2 Instruction Set

(1) Data Transfer Instructions

	Mnemonic	Size	Addressing Mode and Instruction Length (bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@–ERn	@aa	@(d,PC)	@@aa	—		I	H	N	Z	V	C	Normal	Ad-vanced
MOV	MOV.B #xx:8,Rd	B	2									#xx:8→Rd8	—	—	↕	↕	0	—	2	2
	MOV.B Rs,Rd	B		2								Rs8→Rd8	—	—	↕	↕	0	—	2	2
	MOV.B @ERs,Rd	B			2							@ERs→Rd8	—	—	↕	↕	0	—	4	4
	MOV.B @(d:16, ERs), Rd	B				4						@(d:16,ERs)→Rd8	—	—	↕	↕	0	—	6	6
	MOV.B @(d:24,ERs),Rd	B				8						@(d24:,ERs24)→Rd8	—	—	↕	↕	0	—	10	10
	MOV.B @ERs+,Rd	B					2					@ERs→Rd8,ERs32+1→ERs32	—	—	↕	↕	0	—	6	6
	MOV.B @aa:8,Rd	B						2				@aa:8→Rd8	—	—	↕	↕	0	—	4	4
	MOV.B @aa:16,Rd	B						4				@aa:16→Rd8	—	—	↕	↕	0	—	6	6
	MOV.B @aa:24,Rd	B						6				@aa:24→Rd8	—	—	↕	↕	0	—	8	8
	MOV.B Rs,@ERd	B			2							Rs8→@ERd24	—	—	↕	↕	0	—	4	4
	MOV.B Rs,@(d:16,ERd)	B				4						Rd8→@(d:16,ERd)	—	—	↕	↕	0	—	6	6
	MOV.B Rs,@(d:24,ERd)	B				8						Rd8→@(d:24,ERd)	—	—	↕	↕	0	—	10	10
	MOV.B Rs,@–ERd	B					2					ERd32-1→ERd32,Rs8→@ERd	—	—	↕	↕	0	—	6	6
	MOV.B Rs,@aa:8	B						2				Rs8→@aa:8	—	—	↕	↕	0	—	4	4
	MOV.B Rs,@aa:16	B						4				Rs8→@aa:16	—	—	↕	↕	0	—	6	6
	MOV.B Rs,@aa:24	B						6				Rs8→@aa:24	—	—	↕	↕	0	—	8	8
	MOV.W #xx:16,Rd	W	4									#xx:16→Rd16	—	—	↕	↕	0	—	4	4
	MOV.W Rs,Rd	W		2								Rs16→Rd16	—	—	↕	↕	0	—	2	2
	MOV.W @ERs,Rd	W			2							@ERs24→Rd16	—	—	↕	↕	0	—	4	4
	MOV.W @(d:16,ERs),Rd	W				4						@(d:16,ERs)→Rd16	—	—	↕	↕	0	—	6	6
	MOV.W @(d:24,ERs),Rd	W				8						@(d:24,ERs)→Rd16	—	—	↕	↕	0	—	10	10
	MOV.W @ERs+,Rd	W					2					@ERs→Rd16,ERs32+2→@ERd	—	—	↕	↕	0	—	6	6
	MOV.W @aa:16,Rd	W						4				@aa:16→Rd16	—	—	↕	↕	0	—	6	6
	MOV.W @aa:24,Rd	W						6				@aa:24→Rd16	—	—	↕	↕	0	—	8	8
	MOV.W Rs,@ERd	W			2							Rs16→@ERd	—	—	↕	↕	0	—	4	4
	MOV.W Rs,@(d:16,ERd)	W				4						Rs16→@(d:16,ERd)	—	—	↕	↕	0	—	6	6
	MOV.W Rs,@(d:24,ERd)	W				8						Rs16→@(d:24,ERd)	—	—	↕	↕	0	—	8	10
	MOV.W Rs,@–ERd	W					2					ERd32-2→ERd32,Rs16→@ERd24	—	—	↕	↕	0	—	6	6
	MOV.W Rs,@aa:16	W						4				Rs16→@aa:16	—	—	↕	↕	0	—	6	6
	MOV.W Rs,@aa:24	W						6				Rs16→@aa:24	—	—	↕	↕	0	—	8	8
	MOV.L #xx:32,ERd	L	6									#xx:32→ERd32	—	—	↕	↕	0	—	8	6
	MOV.L ERs,ERd	L		2								ERs32→ERd32	—	—	↕	↕	0	—	2	2
	MOV.L @ERs,ERd	L			4							@ERs→ERd32	—	—	↕	↕	0	—	8	8

Mnemonic		Size	Addressing Mode and Instruction Length (bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@–ERn	@aa	@(d,PC)	@@aa	—		I	H	N	Z	V	C	Normal	Ad-vanced
MOV	MOV.L @(d:16,ERs),ERd	L				6						@(d:16,ERs)→ERd32	—	—	↕	↕	0	—	10	10
	MOV.L @(d:24,ERs),ERd	L				10						@(d:24,ERs)→ERd32	—	—	↕	↕	0	—	14	14
	MOV.L @ERs+,ERd	L					4					ERs→ERd32,ERs32+4→@ERs32	—	—	↕	↕	0	—	10	10
	MOV.L @aa:16,ERd	L						6				@aa:16→ERd32	—	—	↕	↕	0	—	10	10
	MOV.L @aa:24,ERd	L						8				@aa:24→ERd32	—	—	↕	↕	0	—	12	12
	MOV.L ERs,@ERd	L			4							ERs32→@ERd24	—	—	↕	↕	0	—	8	8
	MOV.L ERs,@(d:16,ERd)	L				6						ERs32→@(d:16,ERd)	—	—	↕	↕	0	—	10	10
	MOV.L ERs,@(d:24,ERd)	L				10						ERs32→@(d:24,ERd)	—	—	↕	↕	0	—	14	14
	MOV.L ERs,@–ERd	L					4					ERd32-4→ERd32,ERs32→@ERd	—	—	↕	↕	0	—	10	10
	MOV.L ERs,@aa:16	L						6				ERs32→@aa:16	—	—	↕	↕	0	—	10	10
	MOV.L ERs,@aa:24	L						8				ERs32→@aa:24	—	—	↕	↕	0	—	12	12
POP	POP.W Rn	W									2	@SP→Rn16,SP+2→SP	—	—	↕	↕	0	—	6	6
	POP.L ERn	L									4	@SP→ERn32,SP+4→SP	—	—	↕	↕	0	—	8	10
PUSH	PUSH.W Rn	W									2	SP-2→SP,Rn16→@SP	—	—	↕	↕	0	—	6	6
	PUSH.L ERn	L									4	SP-4→SP,ERn32→@SP	—	—	↕	↕	0	—	8	10
MOVFPE	MOVFPE@aa:16,Rd	B						4				@aa:16→Rd (synchronized with E clock)	—	—	↕	↕	0	—	(6)	(6)
MOVTPE	MOVTPE Rs,@aa:16	B						4				Rs→@aa:16 (synchronized with E clock)R	—	—	↕	↕	0	—	(6)	(6)

(2) Arithmetic Operation Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@–ERn	@aa	@(d,PC)	@@aa	—		I	H	N	Z	V	C	Normal	Ad-vanced
ADD	ADD.B #xx:8,Rd	B	2									Rd8+#xx:8→Rd8	—	↕	↕	↕	↕	↕	2	2
	ADD.B Rs,Rd	B		2								Rd8+Rs8→Rd8	—	↕	↕	↕	↕	↕	2	2
	ADD.W #xx:16,Rd	W	4									Rd16+#xx:16→Rd16	—	(1)	↕	↕	↕	↕	4	4
	ADD.W Rs,Rd	W		2								Rd16+Rs16→Rd16	—	(1)	↕	↕	↕	↕	2	2
	ADD.L #xx:32,ERd	L	6									ERd32+#xx:32→ERd32	—	(2)	↕	↕	↕	↕	6	6
	ADD.L ERs,ERd	L		2								ERd32+ERs32→ERd32	—	(2)	↕	↕	↕	↕	2	2
ADDX	ADDX #xx:8,Rd	B	2									Rd8+#xx:8+C→Rd8	—	↕	↕	(3)	↕	↕	2	2
	ADDX Rs,Rd	B		2								Rd8+Rs8+C→Rd8	—	↕	↕	(3)	↕	↕	2	2
ADDS	ADDS.L #1,ERd	L		2								ERd32+1→ERd32	—	—	—	—	—	—	2	2
	ADDS.L #2,ERd	L		2								ERd32+2→ERd32	—	—	—	—	—	—	2	2
	ADDS.L #4,ERd	L		2								ERd32+4→ERd32	—	—	—	—	—	—	2	2
INC	INC.B Rd	B		2								Rd8+1→Rd8	—	—	↕	↕	↕	—	2	2
	INC.W #1,Rd	W		2								Rd16+1→Rd16	—	—	↕	↕	↕	—	2	2
	INC.W #2,Rd	W		2								Rd16+2→Rd16	—	—	↕	↕	↕	—	2	2
	INC.L #1,ERd	L		2								ERd32+1→ERd32	—	—	↕	↕	↕	—	2	2
	INC.L #2,ERd	L		2								ERd32+2→ERd32	—	—	↕	↕	↕	—	2	2
DAA	DAA Rd	B		2								Rd8 decimal adjust →Rd8	—	*	↕	↕	*	↕	2	2
SUB	SUB.B Rs,Rd	B		2								Rd8–Rs8→Rd8	—	↕	↕	↕	↕	↕	2	2
	SUB.W #xx:16,Rd	W	4									Rd16–#xx:16→Rd16	—	(1)	↕	↕	↕	↕	4	4
	SUB.W Rs,Rd	W		2								Rd16–Rs16→Rd16	—	(1)	↕	↕	↕	↕	2	2
	SUB.L #xx:32,ERd	L	6									ERd32–#xx:32→ERd32	—	(2)	↕	↕	↕	↕	6	6
	SUB.L ERs,ERd	L		2								ERd32–ERs32→ERd32	—	(2)	↕	↕	↕	↕	2	2
SUBX	SUBX.B #xx:8,Rd	B	2									Rd8–#xx:8–C→Rd8	—	↕	↕	(3)	↕	↕	2	2
	SUBX.B Rs,Rd	B		2								Rd8–Rs8–C→Rd8	—	↕	↕	(3)	↕	↕	2	2
SUBS	SUBS.L #1,ERd	L		2								Erd32–1→ERd32	—	—	—	—	—	—	2	2
	SUBS.L #2,ERd	L		2								ERd32–2→ERd32	—	—	—	—	—	—	2	2
	SUBS.L #4,ERd	L		2								ERd32–4→ERd32	—	—	—	—	—	—	2	2
DEC	DEC.B Rd	B		2								Rd8–1→Rd8	—	—	↕	↕	↕	—	2	2
	DEC.W #1,Rd	W		2								Rd16–1→Rd16	—	—	↕	↕	↕	—	2	2
	DEC.W #2,Rd	W		2								Rd16–2→Rd16	—	—	↕	↕	↕	—	2	2
	DEC.L #1,ERd	L		2								ERd32–1→ERd32	—	—	↕	↕	↕	—	2	2
	DEC.L #2,ERd	L		2								ERd32–2→ERd32	—	—	↕	↕	↕	—	2	2
DAS	DAS Rd	B		2								Rd8 decimal adjust →Rd8	—	*	↕	↕	*	—	2	2

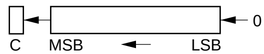
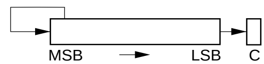
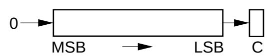
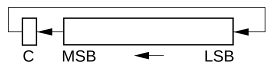
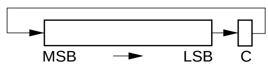
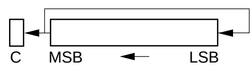
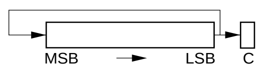
RENESAS

Mnemonic		Size	Addressing Mode and Instruction Length (bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@−ERn	@aa	@(d,PC)	@@aa	—		I	H	N	Z	V	C	Normal	Ad-vanced
NEG	NEG.B Rd	B		2								0–Rd8→Rd8	—	↕	↕	↕	↕	↕	2	2
	NEG.W Rd	W		2								0–Rd16→Rd16	—	↕	↕	↕	↕	↕	2	2
	NEG.L ERd	L		2								0–ERd32-ERd32	—	↕	↕	↕	↕	↕	2	2
CMP	CMP.B #xx:8,Rd	B	2									Rd8–#xx:8	—	↕	↕	↕	↕	↕	2	2
	CMP.B Rs,Rd	B		2								Rd8–Rs8	—	↕	↕	↕	↕	↕	2	2
	CMP.W #xx:16,Rd	W	4									Rd16–#xx:16	—	(1)	↕	↕	↕	↕	4	4
	CMP.W Rs,Rd	W		2								Rd16–Rs16	—	(1)	↕	↕	↕	↕	2	2
	CMP.L #xx:32,ERd	L	6									ERd32–#xx:32	—	(2)	↕	↕	↕	↕	4	6
	CMP.L ERs,ERd	L		2								ERd32–ERs32	—	(2)	↕	↕	↕	↕	2	2
MULXU	MULXU.B Rs,Rd	B		2								Rd8 × Rs8→Rd16 (unsigned operation)	—	—	—	—	—	—	14	14
	MULXU.W Rs,ERd	W		2								Rd16 × Rs16→ERd32 (unsigned operation)	—	—	—	—	—	—	22	22
MULXS	MULXS.B Rs,Rd	B		4								Rd8 × Rs8 → Rd16 (signed operation)	—	—	↕	↕	—	—	16	16
	MULXS.W Rs,ERd	W		4								Rd16 × Rs16 → ERd32 (signed operation)	—	—	↕	↕	—	—	24	24
DIVXU	DIVXU.B Rs,Rd	B		2								Rd16 ÷ Rs8 → Rd16 (RdH: remainder, RdL: quotient) (unsigned operation)	—	—	(6)	(7)	—	—	14	14
	DIVXU.W Rs,ERd	W		2								ERd32 ÷ Rs16 → ERd32 (Ed: remainder, Rd: quotient) (unsigned operation)	—	—	(6)	(7)	—	—	22	22
DIVXS	DIVXS.B Rs,Rd	B		4								Rd16 ÷ Rs8 → Rd16 (RdH: remainder, RdL: quotient) (signed operation)	—	—	(8)	(7)	—	—	16	16
	DIVXS.W Rs,ERd	W		4								ERd32 ÷ Rs16 → ERd32 (Ed: remainder, Rd: quotient) (signed operation)	—	—	(8)	(7)	—	—	24	24
EXTU	EXTU.W Rd	W		2								0 → (<bits 15 to 8> of Rd16)	—	—	0	↕	0	—	2	2
	EXTU.L ERd	L		2								0 → (<bits 31 to 16> of ERd32)	—	—	0	↕	0	—	2	2
EXTS	EXTS.W Rd	W		2								(<bit 7> of Rd16) → (<bits 15 to 8> of Rd16)	—	—	↕	↕	0	—	2	2
	EXTS.L ERd	L		2								(<bit 15> of ERd32) → (<bits 31 to 16> of ERd32)	—	—	↕	↕	0	—	2	2

(3) Logic Operation Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (bytes): #xx	Rn	@ERn	@(d,ERn)	@ERn+/@–ERn	@aa	@(d,PC)	@@aa	—	Operation	Condition Code: I	H	N	Z	V	C	No. of States: Normal	Advanced
AND	AND.B #xx:8,Rd	B	2									Rd8 ∧ #xx:8→Rd8	—	—	↕	↕	0	—	2	2
	AND.B Rs,Rd	B		2								Rd8 ∧ Rs8→Rd8	—	—	↕	↕	0	—	2	2
	AND.W #xx:16,Rd	W	4									Rd16 ∧ #xx:16→Rd16	—	—	↕	↕	0	—	4	4
	AND.W Rs,Rd	W		2								Rd16 ∧ Rs16→Rd16	—	—	↕	↕	0	—	2	2
	AND.L #xx:32,ERd	L	6									ERd32 ∧ #xx:32→ERd32	—	—	↕	↕	0	—	6	6
	AND.L ERs,ERd	L		4								ERd32 ∧ ERs32→ERd32	—	—	↕	↕	0	—	4	4
OR	OR.B #xx:8,Rd	B	2									Rd8 ∨ #xx:8→Rd8	—	—	↕	↕	0	—	2	2
	OR.B Rs,Rd	B		2								Rd8 ∨ Rs8→Rd8	—	—	↕	↕	0	—	2	2
	OR.W #xx:16,Rd	W	4									Rd16 ∨ #xx:16→Rd16	—	—	↕	↕	0	—	4	4
	OR.W Rs,Rd	W		2								Rd16 ∨ Rs16→Rd16	—	—	↕	↕	0	—	2	2
	OR.L #xx:32,ERd	L	6									ERd32 ∨ #xx:32→ERd32	—	—	↕	↕	0	—	6	6
	OR.L ERs,ERd	L		4								ERd32 ∨ ERs32→ERd32	—	—	↕	↕	0	—	4	4
XOR	XOR.B #xx:8,Rd	B	2									Rd8⊕#xx:8→Rd8	—	—	↕	↕	0	—	2	2
	XOR.B Rs,Rd	B		2								Rd8⊕Rs8→Rd8	—	—	↕	↕	0	—	2	2
	XOR.W #xx:16,Rd	W	4									Rd16⊕#xx:16→Rd16	—	—	↕	↕	0	—	4	4
	XOR.W Rs,Rd	W		2								Rd16⊕Rs16→Rd16	—	—	↕	↕	0	—	2	2
	XOR.L #xx:32,ERd	L	6									ERd32⊕#xx:32→ERd32	—	—	↕	↕	0	—	6	6
	XOR.L ERs,ERd	L		4								ERd32⊕ERs32→ERd32	—	—	↕	↕	0	—	4	4
NOT	NOT.B Rd	B		2								¬ Rd8→Rd8	—	—	↕	↕	0	—	2	2
	NOT.W Rd	W		2								¬ Rd16→Rd16	—	—	↕	↕	0	—	2	2
	NOT.L ERd	L		2								¬ Rd32→Rd32	—	—	↕	↕	0	—	2	2

(4) Shift Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (bytes)								Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@−ERn	@aa	@(d,PC)	@@aa	—		I	H	N	Z	V	C	Normal	Ad-vanced
SHAL	SHAL.B Rd	B		2									—	—	↕	↕	↕	↕	2	2
	SHAL.W Rd	W		2									—	—	↕	↕	↕	↕	2	2
	SHAL.L ERd	L		2									—	—	↕	↕	↕	↕	2	2
SHAR	SHAR.B Rd	B		2									—	—	↕	↕	0	↕	2	2
	SHAR.W Rd	W		2									—	—	↕	↕	0	↕	2	2
	SHAR.L ERd	L		2									—	—	↕	↕	0	↕	2	2
SHLL	SHLL.B Rd	B		2									—	—	↕	↕	0	↕	2	2
	SHLL.W Rd	W		2									—	—	↕	↕	0	↕	2	2
	SHLL.L ERd	L		2									—	—	↕	↕	0	↕	2	2
SHLR	SHLR.B Rd	B		2									—	—	↕	↕	0	↕	2	2
	SHLR.W Rd	W		2									—	—	↕	↕	0	↕	2	2
	SHLR.L ERd	L		2									—	—	↕	↕	0	↕	2	2
ROTXL	ROTXL.B Rd	B		2									—	—	↕	↕	0	↕	2	2
	ROTXL.W Rd	W		2									—	—	↕	↕	0	↕	2	2
	ROTXL.L ERd	L		2									—	—	↕	↕	0	↕	2	2
ROTXR	ROTXR.B Rd	B		2									—	—	↕	↕	0	↕	2	2
	ROTXR.W Rd	W		2									—	—	↕	↕	0	↕	2	2
	ROTXR.L ERd	L		2									—	—	↕	↕	0	↕	2	2
ROTL	ROTL.B Rd	B		2									—	—	↕	↕	0	↕	2	2
	ROTL.W Rd	W		2									—	—	↕	↕	0	↕	2	2
	ROTL.L ERd	L		2									—	—	↕	↕	0	↕	2	2
ROTR	ROTR.B Rd	B		2									—	—	↕	↕	0	↕	2	2
	ROTR.W Rd	W		2									—	—	↕	↕	0	↕	2	2
	ROTR.L ERd	L		2									—	—	↕	↕	0	↕	2	2

(5) Bit Manipulation Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@–ERn	@aa	@(d,PC)	@@aa	—		I	H	N	Z	V	C	Normal	Advanced
BSET	BSET #xx:3,Rd	B		2								(#xx:3 of Rd8)←1	—	—	—	—	—	—	2	2
	BSET #xx:3,@ERd	B			4							(#xx:3 of @ERd)←1	—	—	—	—	—	—	8	8
	BSET #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)←1	—	—	—	—	—	—	8	8
	BSET Rn,Rd	B		2								(Rn8 of Rd8)←1	—	—	—	—	—	—	2	2
	BSET Rn,@ERd	B			4							(Rn8 of @ERd)←1	—	—	—	—	—	—	8	8
	BSET Rn,@aa:8	B						4				(Rn8 of @aa:8)←1	—	—	—	—	—	—	8	8
BCLR	BCLR #xx:3,Rd	B		2								(#xx:3 of Rd8)←0	—	—	—	—	—	—	2	2
	BCLR #xx:3,@ERd	B			4							(#xx:3 of @ERd)←0	—	—	—	—	—	—	8	8
	BCLR #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)←0	—	—	—	—	—	—	8	8
	BCLR Rn,Rd	B		2								(Rn8 of Rd8)←0	—	—	—	—	—	—	2	2
	BCLR Rn,@ERd	B			4							(Rn8 of @ERd)←0	—	—	—	—	—	—	8	8
	BCLR Rn,@aa:8	B						4				(Rn8 of @aa:8)←0	—	—	—	—	—	—	8	8
BNOT	BNOT #xx:3,Rd	B		2								(#xx:3 of Rd8)←¬ (#xx:3 of Rd8)	—	—	—	—	—	—	2	2
	BNOT #xx:3,@ERd	B			4							(#xx:3 of @ERd)←¬ (#xx:3 of @ERd)	—	—	—	—	—	—	8	8
	BNOT #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)←¬ (#xx:3 of @aa:8)	—	—	—	—	—	—	8	8
	BNOT Rn,Rd	B		2								(Rn8 of Rd8)←¬ (Rn8 of Rd8)	—	—	—	—	—	—	2	2
	BNOT Rn,@ERd	B			4							(Rn8 of @ERd)←¬ (Rn8 of @ERd)	—	—	—	—	—	—	8	8
	BNOT Rn,@aa:8	B						4				(Rn8 of @aa:8)←¬ (Rn8 of @aa:8)	—	—	—	—	—	—	8	8
BTST	BTST #xx:3,Rd	B		2								(#xx:3 of Rd8)→Z	—	—	—	↕	—	—	2	2
	BTST #xx:3,@ERd	B				4						(#xx:3 of @ERd)→Z	—	—	—	↕	—	—	6	6
	BTST #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)→Z	—	—	—	↕	—	—	6	6
	BTST Rn,Rd	B		2								(Rn8 of Rd8)→Z	—	—	—	↕	—	—	2	2
	BTST Rn,@ERd	B			4							(Rn8 of @ERd)→Z	—	—	—	↕	—	—	6	6
	BTST Rn,@aa:8	B						4				(Rn8 of @aa:8)→Z	—	—	—	↕	—	—	6	6
BLD	BLD #xx:3,Rd	B		2								(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BLD #xx:3,@ERd	B			4							(#xx:3 of @ERd)→C	—	—	—	—	—	↕	6	6
	BLD #xx:3,@aa:8	B						4				(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6
BILD	BILD #xx:3,Rd	B		2								¬ (#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BILD #xx:3,@ERd	B			4							¬ (#xx:3 of @ERd24)→C	—	—	—	—	—	↕	6	6
	BILD #xx:3,@aa:8	B						4				¬ (#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6

Mnemonic		Size	Addressing Mode and Instruction Length (bytes)									Operation	Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@−ERn	@aa	@(d,PC)	@@aa	—		I	H	N	Z	V	C	Normal	Advanced
BST	BST #xx:3,Rd	B		2								C→(#xx:3 of Rd8)	—	—	—	—	—	—	2	2
	BST #xx:3,@ERd	B			4							C→(#xx:3 of @ERd24)	—	—	—	—	—	—	8	8
	BST #xx:3,@aa:8	B						4				C→(#xx:3 of @aa:8)	—	—	—	—	—	—	8	8
BIST	BIST #xx:3,Rd	B		2								/C→(#xx:3 of Rd8)	—	—	—	—	—	—	2	2
	BIST #xx:3,@ERd	B			4							/C→(#xx:3 of @ERd24)	—	—	—	—	—	—	8	8
	BIST #xx:3,@aa:8	B						4				/C→(#xx:3 of @aa:8)	—	—	—	—	—	—	8	8
BAND	BAND #xx:3,Rd	B		2								C∧(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BAND #xx:3,@ERd	B			4							C∧(#xx:3 of @ERd24)→C	—	—	—	—	—	↕	6	6
	BAND #xx:3,@aa:8	B						4				C∧(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6
BIAND	BIAND #xx:3,Rd	B		2								C∧¬ (/#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BIAND #xx:3,@ERd	B			4							C∧¬ (/#xx:3 of @ERd24)→C	—	—	—	—	—	↕	6	6
	BIAND #xx:3,@aa:8	B						4				C∧¬ (/#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6
BOR	BOR #xx:3,Rd	B		2								C ∨ (#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BOR #xx:3,@ERd	B			4							C ∨ (#xx:3 of @ERd24)→C	—	—	—	—	—	↕	6	6
	BOR #xx:3,@aa:8	B						4				C ∨ (#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6
BIOR	BIOR #xx:3,Rd	B		2								C ∨ ~(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BIOR #xx:3,@ERd	B			4							C ∨ ~(#xx:3 of @ERd24)→C	—	—	—	—	—	↕	6	6
	BIOR #xx:3,@aa:8	B						4				C ∨ ~(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6
BXOR	BXOR #xx:3,Rd	B		2								C ⊕ (#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BXOR #xx:3,@ERd	B			4							C ⊕ (#xx:3 of @ERd24)→C	—	—	—	—	—	↕	6	6
	BXOR #xx:3,@aa:8	B						4				C ⊕ (#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6
BIXOR	BIXOR #xx:3,Rd	B		2								C ⊕ ~(#xx:3 of Rd8)→C	—	—	—	—	—	↕	2	2
	BIXOR #xx:3,@ERd	B			4							C ⊕ ~(#xx:3 of @ERd24)→C	—	—	—	—	—	↕	6	6
	BIXOR #xx:3,@aa:8	B						4				C ⊕ ~(#xx:3 of @aa:8)→C	—	—	—	—	—	↕	6	6

(6) Branch Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (bytes): #xx	Rn	@ERn	@(d,ERn)	@ERn+/@−ERn	@aa	@(d,PC)	@@aa	—	Operation	Branch condition	Condition Code: I	H	N	Z	V	C	No. of States: Normal	Advanced
Bcc	BRA d:8(BTd:8)	—							2			if condition is true then PC←PC+d else next;	Always	—	—	—	—	—	—	4	4
	BRA d:16(BTd:16)	—							4					—	—	—	—	—	—	6	6
	BRN d:8(BFd:8)	—							2				Never	—	—	—	—	—	—	4	4
	BRN d:16(BFd:16)	—							4					—	—	—	—	—	—	6	6
	BHI d:8	—							2				C ∨ Z = 0	—	—	—	—	—	—	4	4
	BHI d:16	—							4					—	—	—	—	—	—	6	6
	BLS d:8	—							2				C ∨ Z = 1	—	—	—	—	—	—	4	4
	BLS d:16	—							4					—	—	—	—	—	—	6	6
	BCC d:8(BHS d:8)	—							2				C = 0	—	—	—	—	—	—	4	4
	BCC d:16(BHS d:16)	—							4					—	—	—	—	—	—	6	6
	BCS d:8(BLO d:8)	—							2				C = 1	—	—	—	—	—	—	4	4
	BCS d:16(BLO d:16)	—							4					—	—	—	—	—	—	6	6
	BNE d:8	—							2				Z = 0	—	—	—	—	—	—	4	4
	BNE d:16	—							4					—	—	—	—	—	—	6	6
	BEQ d:8	—							2				Z = 1	—	—	—	—	—	—	4	4
	BEQ d:16	—							4					—	—	—	—	—	—	6	6
	BVC d:8	—							2				V = 0	—	—	—	—	—	—	4	4
	BVC d:16	—							4					—	—	—	—	—	—	6	6
	BVS d:8	—							2				V = 1	—	—	—	—	—	—	4	4
	BVS d:16	—							4					—	—	—	—	—	—	6	6
	BPL d:8	—							2				N = 0	—	—	—	—	—	—	4	4
	BPL d:16	—							4					—	—	—	—	—	—	6	6
	BMI d:8	—							2				N = 1	—	—	—	—	—	—	4	4
	BMI d:16	—							4					—	—	—	—	—	—	6	6
	BGE d:8	—							2				N ⊕ V = 0	—	—	—	—	—	—	4	4
	BGE d:16	—							4					—	—	—	—	—	—	6	6
	BLT d:8	—							2				N ⊕ V = 1	—	—	—	—	—	—	4	4
	BLT d:16	—							4					—	—	—	—	—	—	6	6
	BGT d:8	—							2				Z ∨ (N ⊕ V) = 0	—	—	—	—	—	—	4	4
	BGT d:16	—							4					—	—	—	—	—	—	6	6
	BLE d:8	—							2				Z ∨ (N ⊕ V) = 1	—	—	—	—	—	—	4	4
	BLE d:16	—							4					—	—	—	—	—	—	6	6

Mnemonic		Size	Addressing Mode and Instruction Length (bytes)									Operation		Condition Code						No. of States	
			#xx	Rn	@ERn	@(d,ERn)	@ERn+/@−ERn	@aa	@(d,PC)	@@aa	—		Branch condition	I	H	N	Z	V	C	Normal	Ad-vanced
JMP	JMP @ERn	—			2							PC←ERn		—	—	—	—	—	—	4	4
	JMP @aa:24	—						4				PC←aa:24		—	—	—	—	—	—	6	6
	JMP @@aa:8	—								2		PC←@aa:8		—	—	—	—	—	—	8	10
BSR	BSR d:8	—							2			PC→@-SP, PC←PC+d:8		—	—	—	—	—	—	6	8
	BSR d:16	—							4			PC→@-SP, PC←PC+d:16		—	—	—	—	—	—	8	10
JSR	JSR @ERn	—			2							PC→@-SP, PC←ERn		—	—	—	—	—	—	6	8
	JSR @aa:24	—						4				PC→@-SP, PC←aa:24		—	—	—	—	—	—	8	10
	JSR @@aa:8	—								2		PC→@-SP, PC←@aa:8		—	—	—	—	—	—	8	12
RTS	RTS	—									2	PC←@SP+		—	—	—	—	—	—	8	10

(7) System Control Instructions

Mnemonic		Size	Addressing Mode and Instruction Length (bytes) #xx	Rn	@ERn	@(d,ERn)	@ERn+/@–ERn	@aa	@(d,PC)	@@aa	—	Operation	Condition Code I	H	N	Z	V	C	No. of States Normal	Advanced
TRAPA	TRAPA #x:2	—									2	PC→ @–SP, CCR→@–SP, <vector> → PC	(1)	—	—	—	—	—	14	14
RTE	RTE	—										CCR←@SP+,PC←@SP+	↕	↕	↕	↕	↕	↕	10	10
SLEEP	SLEEP	—										Transition to power-down state	—	—	—	—	—	—	2	2
LDC	LDC #xx:8,CCR	B	2									#xx:8→CCR	↕	↕	↕	↕	↕	↕	2	2
	LDC Rs,CCR	B		2								Rs8→CCR	↕	↕	↕	↕	↕	↕	2	2
	LDC @ERs,CCR	W			4							@ERs→CCR	↕	↕	↕	↕	↕	↕	6	6
	LDC @(d:16,ERs),CCR	W				6						@(d:16,ERs)→CCR	↕	↕	↕	↕	↕	↕	8	8
	LDC @(d:16,ERs),CCR	W				10						@(d:24,ERs)→CCR	↕	↕	↕	↕	↕	↕	12	12
	LDC @ERs+,CCR	W					4					@ERs→CCR,ERs32+2→ERs32	↕	↕	↕	↕	↕	↕	8	8
	LDC @aa:16,CCR	W						6				@aa:16→CCR	↕	↕	↕	↕	↕	↕	8	8
	LDC @aa:24,CCR	W						8				@aa:24→CCR	↕	↕	↕	↕	↕	↕	10	10
STC	STC CCR,Rd	B		2								CCR→Rd8	—	—	—	—	—	—	2	2
	STC CCR,@ERd	W			4							CCR→@ERd	—	—	—	—	—	—	6	6
	STC CCR,@(d:16,ERs)	W				6						CCR→@(d:16,ERs24)	—	—	—	—	—	—	8	8
	STC CCR,@(d:24,ERs)	W				10						CCR→@(d:24,ERs24)	—	—	—	—	—	—	12	12
	STC CCR,@–ERs	W					4					ERd32-2→ERd24,CCR→@ERd24	—	—	—	—	—	—	8	8
	STC CCR,@aa:16	W						6				CCR→@aa:16	—	—	—	—	—	—	8	8
	STC CCR,@aa:24	W						8				CCR→@aa:24	—	—	—	—	—	—	10	10
ANDC	ANDC #xx:8,CCR	B	2									CCR ∧#xx:8→CCR	↕	↕	↕	↕	↕	↕	2	2
ORC	ORC #xx:8,CCR	B	2									CCR V#xx:8→CCR	↕	↕	↕	↕	↕	↕	2	2
XORC	XORC #xx:8,CCR	B	2									CCR⊕#xx:8→CCR	↕	↕	↕	↕	↕	↕	2	2
NOP	NOP	—									2	PC←PC+2	—	↕	—	—	—	—	2	2

(8) Block Transfer Instructions

Mnemonic		Size	#xx	Rn	@ERn	@(d,ERn)	@ERn+/@–ERn	@aa	@(d,PC)	@@aa	—	Operation	I	H	N	Z	V	C	No. of States Normal	No. of States Advanced
EEPMOV	EEPMOV.B	—									4	if R4L ≠ 0 Repeat @R5→@R6 R5+1→R5 R6+1→R6 R4L–1→R4L Until R4L = 0 else next;	—	—	—	—	—	—	8+4n*2	8+4n*2
	EEPMOV.W	—									4	if R4 ≠ 0 Repeat @R5→@R6 R5+1→R5 R6+1→R6 R4L–1→R4L Until R4 = 0 else next;	—	—	—	—	—	—	8+4n*2	8+4n*2

Notes: 1. The number of states is the number of states required for execution when the instruction and its operands are located in on-chip memory. For other cases see section 2.6, Number of States Required for Execution.

2. n is the value set in register R4L or R4.

(1) Set to 1 when a carry or borrow occurs at bit 11; otherwise cleared to 0.

(2) Set to 1 when a carry or borrow occurs at bit 27; otherwise cleared to 0.

(3) Retains its previous value when the result is zero; otherwise cleared to 0.

(4) Set to 1 when the adjustment produces a carry; otherwise retains its previous value.

(5) The number of states required for execution of an instruction that transfers data in synchronization with the E clock is variable.

(6) Set to 1 when the divisor is negative; otherwise cleared to 0.

(7) Set to 1 when the divisor is zero; otherwise cleared to 0.

(8) Set to 1 when the quotient is negative; otherwise cleared to 0.

2.4 Instruction Codes

Table 2.3 Instruction Codes

Instruction	Mnemonic	Size	1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
ADD	ADD.B #xx:8,Rd	B	8 rd	IMM								
	ADD.B Rs,Rd	B	0 8	rs rd								
	ADD.W #xx:16,Rd	W	7 9	1 rd	IMM							
	ADD.W Rs,Rd	W	0 9	rs rd								
	ADD.L #xx:32,ERd	L	7 A	1 0:erd	IMM							
	ADD.L ERs,ERd	L	0 A	1:ers 0:erd								
ADDS	ADDS #1,ERd	L	0 B	0 0:erd								
	ADDS #2,ERd	L	0 B	8 0:erd								
	ADDS #4,ERd	L	0 B	9 0:erd								
ADDX	ADDX #xx:8,Rd	B	9 rd	IMM								
	ADDX Rs,Rd	B	0 E	rs rd								
AND	AND.B #xx:8,Rd	B	E rd	IMM								
	AND.B Rs,Rd	B	1 6	rs rd								
	AND.W #xx:16,Rd	W	7 9	6 rd	IMM							
	AND.W Rs,Rd	W	6 6	rs rd								
	AND.L #xx:32,ERd	L	7 A	6 0:erd	IMM							
	AND.L ERs,ERd	L	0 1	F 0	6 6	0:ers 0:erd						
ANDC	ANDC #xx:8,CCR	B	0 6	IMM								
BAND	BAND #xx:3,Rd	B	7 6	0:IMM rd								
	BAND #xx:3,@ERd	B	7 C	0:erd 0	7 6	0:IMM 0						
	BAND #xx:3,@aa:8	B	7 E	abs	7 6	0:IMM 0						
Bcc	BRA d:8 (BT d:8)	—	4 0	disp								
	BRA d:16 (BT d:16)	—	5 8	0 0	disp							
	BRN d:8 (BF d:8)	—	4 1	disp								
	BRN d:16 (BF d:16)	—	5 8	1 0	disp							
	BHI d:8	—	4 2	disp								
	BHI d:16	—	5 8	2 0	disp							
	BLS d:8	—	4 3	disp								
	BLS d:16	—	5 8	3 0	disp							
	BCC d:8 (BHS d:8)	—	4 4	disp								
	BCC d:16 (BHS d:16)	—	5 8	4 0	disp							
	BCS d:8 (BLO d:8)	—	4 5	disp								

RENESAS

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
Bcc	BCS d:16 (BLO d:16)	—	5 8	5 0	disp							
	BNE d:8	—	4 6	disp								
	BNE d:16	—	5 8	6 0	disp							
	BEQ d:8	—	4 7	disp								
	BEQ d:16	—	5 8	7 0	disp							
	BVC d:8	—	4 8	disp								
	BVC d:16	—	5 8	8 0	disp							
	BVS d:8	—	4 9	disp								
	BVS d:16	—	5 8	9 0	disp							
	BPL d:8	—	4 A	disp								
	BPL d:16	—	5 8	A 0	disp							
	BMI d:8	—	4 B	disp								
	BMI d:16	—	5 8	B 0	disp							
	BGE d:8	—	4 C	disp								
	BGE d:16	—	5 8	C 0	disp							
	BLT d:8	—	4 D	disp								
	BLT d:16	—	5 8	D 0	disp							
	BGT d:8	—	4 E	disp								
	BGT d:16	—	5 8	E 0	disp							
	BLE d:8	—	4 F	disp								
	BLE d:16	—	5 8	F 0	disp							
BCLR	BCLR #xx:3,Rd	B	7 2	0 IMM rd								
	BCLR #xx:3,@ERd	B	7 D	0 erd 0	7 2	0 IMM 0						
	BCLR #xx:3,@aa:8	B	7 F	abs	7 2	0 IMM 0						
	BCLR Rn,Rd	B	6 2	rn rd								
	BCLR Rn,@ERd	B	7 D	0 erd 0	6 2	rn 0						
	BCLR Rn,@aa:8	B	7 F	abs	6 2	rn 0						
BIAND	BIAND #xx:3,Rd	B	7 6	1 IMM rd								
	BIAND #xx:3,@ERd	B	7 C	0 erd 0	7 6	1 IMM 0						
	BIAND #xx:3,@aa:8	B	7 E	abs	7 6	1 IMM 0						
BILD	BILD #xx:3,Rd	B	7 7	1 IMM rd								
	BILD #xx:3,@ERd	B	7 C	0 erd 0	7 7	1 IMM 0						
	BILD #xx:3,@aa:8	B	7 E	abs	7 7	1 IMM 0						

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
BIOR	BIOR #xx:3,Rd	B	7 4	1 IMM rd								
	BIOR #xx:3,@ERd	B	7 C	0 erd 0	7 4	1 IMM 0						
	BIOR #xx:3,@aa:8	B	7 E	abs	7 4	1 IMM 0						
BIST	BIST #xx:3,Rd	B	6 7	1 IMM rd								
	BIST #xx:3,@ERd	B	7 D	0 erd 0	6 7	1 IMM 0						
	BIST #xx:3,@aa:8	B	7 F	abs	6 7	1 IMM 0						
BIXOR	BIXOR #xx:3,Rd	B	7 5	1 IMM rd								
	BIXOR #xx:3,@ERd	B	7 C	0 erd 0	7 5	1 IMM 0						
	BIXOR #xx:3,@aa:8	B	7 E	abs	7 5	1 IMM 0						
BLD	BLD #xx:3,Rd	B	7 7	0 IMM rd								
	BLD #xx:3,@ERd	B	7 C	0 erd 0	7 7	0 IMM 0						
	BLD #xx:3,@aa:8	B	7 E	abs	7 7	0 IMM 0						
BNOT	BNOT #xx:3,Rd	B	7 1	0 IMM rd								
	BNOT #xx:3,@ERd	B	7 D	0 erd 0	7 1	0 IMM 0						
	BNOT #xx:3,@aa:8	B	7 F	abs	7 1	0 IMM 0						
	BNOT Rn,Rd	B	6 1	rn rd								
	BNOT Rn,@ERd	B	7 D	0 erd 0	6 1	rn 0						
	BNOT Rn,@aa:8	B	7 F	abs	6 1	rn 0						
BOR	BOR #xx:3,Rd	B	7 4	0 IMM rd								
	BOR #xx:3,@ERd	B	7 C	0 erd 0	7 4	0 IMM 0						
	BOR #xx:3,@aa:8	B	7 E	abs	7 4	0 IMM 0						
BSET	BSET #xx:3,Rd	B	7 0	0 IMM rd								
	BSET #xx:3,@ERd	B	7 D	0 erd 0	7 0	0 IMM 0						
	BSET #xx:3,@aa:8	B	7 F	abs	7 0	0 IMM 0						
	BSET Rn,Rd	B	6 0	rn rd								
	BSET Rn,@ERd	B	7 D	0 erd 0	6 0	rn 0						
	BSET Rn,@aa:8	B	7 F	abs	6 0	rn 0						
BSR	BSR d:8	—	5 5	disp								
	BSR d:16	—	5 C	0 0	disp							
BST	BST #xx:3,Rd	B	6 7	0 IMM rd								
	BST #xx:3,@ERd	B	7 D	0 erd 0	6 7	0 IMM 0						
	BST #xx:3,@aa:8	B	7 F	abs	6 7	0 IMM 0						

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
BTST	BTST #xx:3,Rd	B	7 : 3	0 IMM : rd								
	BTST #xx:3,@ERd	B	7 : C	0 erd : 0	7 : 3	0 IMM : 0						
	BTST #xx:3,@aa:8	B	7 : E	abs	7 : 3	0 IMM : 0						
	BTST Rn,Rd	B	6 : 3	rn : rd								
	BTST Rn,@ERd	B	7 : C	0 erd : 0	6 : 3	rn : 0						
	BTST Rn,@aa:8	B	7 : E	abs	6 : 3	rn : 0						
BXOR	BXOR #xx:3,Rd	B	7 : 5	0 IMM : rd								
	BXOR #xx:3,@ERd	B	7 : C	0 erd : 0	7 : 5	0 IMM : 0						
	BXOR #xx:3,@aa:8	B	7 : E	abs	7 : 5	0 IMM : 0						
CMP	CMP.B #xx:8,Rd	B	A : rd	IMM								
	CMP.B Rs,Rd	B	1 : C	rs : rd								
	CMP.W #xx:16,Rd	W	7 : 9	2 : rd	IMM							
	CMP.W Rs,Rd	W	1 : D	rs : rd								
	CMP.L #xx:32,ERd	L	7 : A	2 : 0 erd	IMM							
	CMP.L ERs,ERd	L	1 : F	1 ers : 0 erd								
DAA	DAA Rd	B	0 : F	0 : rd								
DAS	DAS Rd	B	1 : F	0 : rd								
DEC	DEC.B Rd	B	1 : A	0 : rd								
	DEC.W #1,Rd	W	1 : B	5 : rd								
	DEC.W #2,Rd	W	1 : B	D : rd								
	DEC.L #1,ERd	L	1 : B	7 : 0 erd								
	DEC.L #2,ERd	L	1 : B	F : 0 erd								
DIVXS	DIVXS.B Rs,Rd	B	0 : 1	D : 0	5 : 1	rs : rd						
	DIVXS.W Rs,ERd	W	0 : 1	D : 0	5 : 3	rs : 0 erd						
DIVXU	DIVXU.B Rs,Rd	B	5 : 1	rs : rd								
	DIVXU.W Rs,ERd	W	5 : 3	rs : 0 erd								
EEPMOV	EEPMOV.B	—	7 : B	5 : C	5 : 9	8 : F						
	EEPMOV.W	—	7 : B	D : 4	5 : 9	8 : F						
EXTS	EXTS.W Rd	W	1 : 7	D : rd								
	EXTS.L ERd	L	1 : 7	F : 0 erd								
EXTU	EXTU.W Rd	W	1 : 7	5 : rd								
	EXTU.L ERd	L	1 : 7	7 : 0 erd								
INC	INC.B Rd	B	0 : A	0 : rd								
	INC.W #1,Rd	W	0 : B	5 : rd								
	INC.W #2,Rd	W	0 : B	D : rd								

Instruction	Mnemonic	Size	Instruction Format 1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
INC	INC.L #1,ERd	L	0 B	7 0 erd								
	INC.L #2,ERd	L	0 B	F 0 erd								
JMP	JMP @ERn	—	5 9	0 ern 0								
	JMP @aa:24	—	5 A	abs								
	JMP @@aa:8	—	5 B	abs								
JSR	JSR @ERn	—	5 D	0 ern 0								
	JSR @aa:24	—	5 E	abs								
	JSR @@aa:8	—	5 F	abs								
LDC	LDC #xx:8,CCR	B	0 7	IMM								
	LDC Rs,CCR	B	0 3	0 rs								
	LDC @ERs,CCR	W	0 1	4 0	6 9	0 ers 0						
	LDC @(d:16,ERs),CCR	W	0 1	4 0	6 F	0 ers 0	disp					
	LDC @(d:24,ERs),CCR	W	0 1	4 0	7 8	0 ers 0	6 B	2 0	0 0	disp		
	LDC @ERs+,CCR	W	0 1	4 0	6 D	0 ers 0						
	LDC @aa:16,CCR	W	0 1	4 0	6 B	0 0	abs					
	LDC @aa:24,CCR	W	0 1	4 0	6 B	2 0	0 0	abs				
MOV	MOV.B #xx:8,Rd	B	F rd	IMM								
	MOV.B Rs,Rd	B	0 C	rs rd								
	MOV.B @ERs,Rd	B	6 8	0 ers rd								
	MOV.B @(d:16,ERs),Rd	B	6 E	0 ers rd	disp							
	MOV.B @(d:24,ERs),Rd	B	7 8	0 ers 0	6 A	2 rd	0 0	disp				
	MOV.B @ERs+,Rd	B	6 C	0 ers rd								
	MOV.B @aa:8,Rd	B	2 rd	abs								
	MOV.B @aa:16,Rd	B	6 A	0 rd	abs							
	MOV.B @aa:24,Rd	B	6 A	2 rd	0 0	abs						
	MOV.B Rs,@ERd	B	6 8	1 erd rs								
	MOV.B Rs,@(d:16,ERd)	B	6 E	1 erd rs	disp							
	MOV.B Rs,@(d:24,ERd)	B	7 8	0 erd 0	6 A	A rs	0 0	disp				
	MOV.B Rs,@–ERd	B	6 C	1 erd rs								
	MOV.B Rs,@aa:8	B	3 rs	abs								
	MOV.B Rs,@aa:16	B	6 A	8 rs	abs							
	MOV.B Rs,@aa:24	B	6 A	A rs	0 0	abs						
	MOV.W #xx:16,Rd	W	7 9	0 rd	IMM							
	MOV.W Rs,Rd	W	0 D	rs rd								
	MOV.W @ERs,Rd	W	6 9	0 ers rd								

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
MOV	MOV.W @(d:16,ERs),Rd	W	6 F	0 ers rd	disp							
	MOV.W @(d:24,ERs),Rd	W	7 8	0 ers 0	6 B	2 rd	0 0	disp				
	MOV.W @ERs+,Rd	W	6 D	0 ers rd								
	MOV.W @aa:16,Rd	W	6 B	0 rd	abs							
	MOV.W @aa:24,Rd	W	6 B	2 rd	0 0	abs						
	MOV.W Rs,@ERd	W	6 9	1 erd rs								
	MOV.W Rs,@(d:16,ERd)	W	6 F	1 erd rs	disp							
	MOV.W Rs,@(d:24,ERd)	W	7 8	1 erd 0	6 B	A rs	0 0	disp				
	MOV.W Rs,@–ERd	W	6 D	1 erd rs								
	MOV.W Rs,@aa:16	W	6 B	8 rs	abs							
	MOV.W Rs,@aa:24	W	6 B	A rs	0 0	abs						
	MOV.L #xx:32,Rd	L	7 A	0 0 erd	IMM							
	MOV.L ERs,ERd	L	0 F	1 ers 0 erd								
	MOV.L @ERs,ERd	L	0 1	0 0	6 9	0 ers 0 erd						
	MOV.L @(d:16,ERs),ERd	L	0 1	0 0	6 F	0 ers 0 erd	disp					
	MOV.L @(d:24,ERs),ERd	L	0 1	0 0	7 8	0 ers 0	6 B	2 0 erd	0 0	disp		
	MOV.L @ERs+,ERd	L	0 1	0 0	6 D	0 ers 0 erd						
	MOV.L @aa:16,ERd	L	0 1	0 0	6 B	0 0 erd	abs					
	MOV.L @aa:24,ERd	L	0 1	0 0	6 B	2 0 erd	0 0	abs				
	MOV.L ERs,@ERd	L	0 1	0 0	6 9	1 erd 0 ers						
	MOV.L ERs,@(d:16,ERd)	L	0 1	0 0	6 F	1 erd 0 ers	disp					
	MOV.L ERs,@(d:24,ERd)	L	0 1	0 0	7 8	0 erd 0	6 B	A 0 ers	0 0	disp		
	MOV.L ERs,@–ERd	L	0 1	0 0	6 D	1 erd 0 ers						
	MOV.L ERs,@aa:16	L	0 1	0 0	6 B	8 0 ers	abs					
	MOV.L ERs,@aa:24	L	0 1	0 0	6 B	A 0 ers	0 0	abs				
MOVFPE	MOVFPE @aa:16,Rd	B	6 A	4 rd	abs							
MOVTPE	MOVTPE Rs,@aa:16	B	6 A	C rs	abs							
MULXS	MULXS.B Rs,Rd	B	0 1	C 0	5 0	rs rd						
	MULXS.W Rs,ERd	W	0 1	C 0	5 2	rs 0 erd						
MULXU	MULXU.B Rs,Rd	B	5 0	rs rd								
	MULXU.W Rs,ERd	W	5 2	rs 0 erd								
NEG	NEG.B Rd	B	1 7	8 rd								
	NEG.W Rd	W	1 7	9 rd								
	NEG.L ERd	L	1 7	B 0 erd								
NOP	NOP	—	0 0	0 0								

Instruction	Mnemonic	Size	Instruction Format									
			1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
NOT	NOT.B Rd	B	1 7	0 rd								
	NOT.W Rd	W	1 7	1 rd								
	NOT.L ERd	L	1 7	3 0 erd								
OR	OR.B #xx:8,Rd	B	C rd	IMM								
	OR.B Rs,Rd	B	1 4	rs rd								
	OR.W #xx:16,Rd	W	7 9	4 rd	IMM							
	OR.W Rs,Rd	W	6 4	rs rd								
	OR.L #xx:32,ERd	L	7 A	4 0 erd	IMM							
	OR.L ERs,ERd	L	0 1	F 0	6 4	0 ers 0 ers						
ORC	ORC #xx:8,CCR	B	0 4	IMM								
POP	POP.W Rn	W	6 D	7 rn								
	POP.L ERn	L	0 1	0 0	6 D	7 0 ern						
PUSH	PUSH.W Rn	W	6 D	F rn								
	PUSH.L ERn	L	0 1	0 0	6 D	F 0 ern						
ROTL	ROTL.B Rd	B	1 2	8 rd								
	ROTL.W Rd	W	1 2	9 rd								
	ROTL.L ERd	L	1 2	B 0 erd								
ROTR	ROTR.B Rd	B	1 3	8 rd								
	ROTR.W Rd	W	1 3	9 rd								
	ROTR.L ERd	L	1 3	B 0 erd								
ROTXL	ROTXL.B Rd	B	1 2	0 rd								
	ROTXL.W Rd	W	1 2	1 rd								
	ROTXL.L ERd	L	1 2	3 0 erd								
ROTXR	ROTXR.B Rd	B	1 3	0 rd								
	ROTXR.W Rd	W	1 3	1 rd								
	ROTXR.L ERd	L	1 3	3 0 erd								
RTE	RTE	—	5 6	7 0								
RTS	RTS	—	5 4	7 0								
SHAL	SHAL.B Rd	B	1 0	8 rd								
	SHAL.W Rd	W	1 0	9 rd								
	SHAL.L ERd	L	1 0	B 0 erd								
SHAR	SHAR.B Rd	B	1 1	8 rd								
	SHAR.W Rd	W	1 1	9 rd								
	SHAR.L ERd	L	1 1	B 0 erd								

Instruction	Mnemonic	Size	Instruction Format 1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte	9th byte	10th byte
SHLL	SHLL.B Rd	B	1 0	0 rd								
	SHLL.W Rd	W	1 0	1 rd								
	SHLL.L ERd	L	1 0	3 0 erd								
SHLR	SHLR.B Rd	B	1 1	0 rd								
	SHLR.W Rd	W	1 1	1 rd								
	SHLR.L ERd	L	1 1	3 0 erd								
SLEEP	SLEEP	—	0 1	8 0								
STC	STC CCR,Rd	B	0 2	0 rd								
	STC CCR,@ERd	W	0 1	4 0	6 9	1 erd 0						
	STC CCR,@(d:16,ERd)	W	0 1	4 0	6 F	1 erd 0	disp					
	STC CCR,@(d:24,ERd)	W	0 1	4 0	7 8	0 erd 0	6 B	A 0	0 0	disp		
	STC CCR,@–ERd	W	0 1	4 0	6 D	1 erd 0						
	STC CCR,@aa:16	W	0 1	4 0	6 B	8 0	abs					
	STC CCR,@aa:24R	W	0 1	4 0	6 B	A 0	0 0	abs				
SUB	SUB.B Rs,Rd	B	1 8	rs rd								
	SUB.W #xx:16,Rd	W	7 9	3 rd	IMM							
	SUB.W Rs,Rd	W	1 9	rs rd								
	SUB.L #xx:32,ERd	L	7 A	3 0 erd	IMM							
	SUB.L ERs,ERd	L	1 A	1 ers 0 erd								
SUBS	SUBS #1,ERd	L	1 B	0 0 erd								
	SUBS #2,ERd	L	1 B	8 0 erd								
	SUBS #4,ERd	L	1 B	9 0 erd								
SUBX	SUBX #xx:8,Rd	B	B rd	IMM								
	SUBX Rs,Rd	B	1 E	rs rd								
TRAPA	TRAPA #x:2	—	5 7	00 IMM 0								
XOR	XOR.B #xx:8,Rd	B	D rd	IMM								
	XOR.B Rs,Rd	B	1 5	rs rd								
	XOR.W #xx:16,Rd	W	7 9	5 rd	IMM							
	XOR.W Rs,Rd	W	6 5	rs rd								
	XOR.L #xx:32,ERd	L	7 A	4 0 erd	IMM							
	XOR.L ERs,ERd	L	0 1	F 0	6 5	0 ers 0 erd						
XORC	XORC #xx:8,CCR	B	0 5	IMM								

Legend:

IMM: Immediate data (2, 3, 8, 16, or 32 bits)

abs: Absolute address (8, 16, or 24 bits)

disp: Displacement (8, 16, or 24 bits)

rs, rd, rn: Register field (4 bits specifying an 8-bit or 16-bit register. rs corresponds to operand symbols such as Rs, rd corresponds to operand symbols such as Rd, and rn corresponds to the operand symbol Rn.)

ers, erd, ern: Register field (3 bits specifying a 32-bit register. ers corresponds to operand symbols such as ERs, erd corresponds to operand symbols such as ERd, and ern corresponds to the operand symbol ERn.)

The register fields specify general registers as follows.

Address Register 32-bit Register		16-bit Register		8-bit Register	
Register Field	**General Register**	**Register Field**	**General Register**	**Register Field**	**General Register**
000	ER0	0000	R0	0000	R0H
001	ER1	0001	R1	0001	R1H
⋮	⋮	⋮	⋮	⋮	⋮
111	ER7	0111	R7	0111	R7H
		1000	E0	1000	R0L
		1001	E1	1001	R1L
		⋮	⋮	⋮	⋮
		1111	E7	1111	R7L

2.5 Operation Code Map

Tables 2.4 to 2.6 show an operation code map.

Table 2.4 Operation Code Map (1)

Operation Code:

1st byte		2nd byte	
AH	AL	BH	BL

Diagonally split cell: upper-left — Instruction when most significant bit of BH is 0.; lower-right — Instruction when most significant bit of BH is 1.

AH \ AL	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	Table 2.5	STC	LDC	ORG	XORG	ANDC	LDC	ADD		Table 2.5	Table 2.5	MOV		ADDX	Table 2.5
1	Table 2.5	Table 2.5	Table 2.5	Table 2.5	OR.B	XOR.B	AND.B	Table 2.5	SUB.B	SUB.W	Table 2.5	Table 2.5	CMP		SUBX	Table 2.5
2	MOV.B															
3																
4	BRA	BRN	BHI	BLS	BCC	BCS	BNE	BEQ	BVC	BVS	BPL	BMI	BGE	BLT	BGT	BLE
5	MULXU	DIVXU	MULXU	DIVXU	RTS	BSR	RTE	TRAPA	Table 2.5	JMP			BSR	JSR		
6	BSET	BNOT	BCLR	BTST	OR.W	XOR.W	AND.W	BST / BIST	MOV							
7					BOR / BIOR	BXOR / BIXOR	BAND / BIAND	BLD / BILD	MOV	Table 2.5	Table 2.5	EEPMOV	Table 2.6			
8	ADD															
9	ADDX															
A	CMP															
B	SUBX															
C	OR															
D	XOR															
E	AND															
F	MOV															

Table 2.5 Operation Code Map (2)

Operation Code:

1st byte		2nd byte	
AH	AL	BH	BL

AH AL \ BH	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01	MOV				LDC / STC				SLEEP				Table 2.6	Table 2.6		Table 2.6
0A	INC								ADD							
0B	ADDS					INC		INC	ADDS	ADDS				INC		INC
0F	DAA								MOV							
10	SHLL			SHLL					SHAL			SHAL				
11	SHLR			SHLR					SHAR			SHAR				
12	ROTXL			ROTXL					ROTL			ROTL				
13	ROTXR			ROTXR					ROTR			ROTR				
17	NOT			NOT		EXTU		EXTU	NEG			NEG		EXTS		EXTS
1A	DEC								SUB							
1B	SUBS					DEC		DEC	SUB					DEC		DEC
1F	DAS								CMP							
58	BRA	BRN	BHI	BLS	BCC	BCS	BNE	BEQ	BVC	BVS	BPL	BMI	BGE	BLT	BGT	BLE
79	MOV	ADD	CMP	SUB	OR	XOR	AND									
7A	MOV	ADD	CMP	SUB	OR	XOR	AND									

Table 2.6   Operation Code Map (3)

Operation Code:

1st byte		2nd byte		3rd byte		4th byte	
AH	AL	BH	BL	CH	CL	DH	DL

Split cells (X / Y): X = Instruction when most significant bit of DH is 0. Y = Instruction when most significant bit of DH is 1.

AHALBHBLCH \ CL	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01C05	MULXS		MULXS													
01D05		DIVXS		DIVXS												
01F06					OR	XOR	AND									
7Cr06*1				BTST												
7Cr07*1				BTST	BOR / BIOR	BXOR / BIXOR	BAND / BIAND	BID / BILD								
7Dr06*1	BSET	BNOT	BCLR					BST / BIST								
7Dr07*1	BSET	BNOT	BCLR													
7Eaa6*2				BTST												
7Eaa7*2				BTST	BOR / BIOR	BXOR / BIXOR	BAND / BIAND	BID / BILD								
7Faa6*2	BSET	BNOT	BCLR					BST / BIST								
7Faa7*2	BSET	BNOT	BCLR													

Notes: 1. r is a register field.
2. aa is an absolute address field.

2.6 Number of States Required for Instruction Execution

The tables in this section can be used to calculate the number of states required for instruction execution by the H8/300H CPU. Table 2.8 indicates the number of instruction fetch, data read/write, and other cycles occurring in each instruction. Table 2.7 indicates the number of states required for each size. The number of states required for execution of an instruction can be calculated from these two tables as follows:

$$\text{Execution states} = I \times S_I + J \times S_J + K \times S_K + L \times S_K + M \times S_M + N \times S_N$$

Examples: Advanced mode, stack located in external memory, on-chip supporting modules accessed with 8-bit bus width, external devices accessed in three states with one wait state and 16-bit bus width.

1. BSET #0, @FFFFC7:8

   From table 2.8:

   I = L = 2, J = K = M = N= 0

   From table 2.7:

   SI = 4, SL = 3

   Number of states required for execution = $2 \times 4 + 2 \times 3 = 14$

2. JSR @@30

   From table 2.8:

   I = J = K = 2, L = M = N = 0

   From table 2.7:

   SI = SJ = SK = 4

   Number of states required for execution = $2 \times 4 + 2 \times 4 + 2 \times 4 = 24$

Table 2.7 Number of States per Cycle

<table>
<tr><th colspan="2" rowspan="3">Cycle</th><th rowspan="3">On-Chip Memory</th><th colspan="6">Access Conditions</th></tr>
<tr><th colspan="2">On-Chip Supporting Module</th><th colspan="2">8-Bit Bus</th><th colspan="2">16-Bit Bus</th></tr>
<tr><th>8-Bit Bus</th><th>16-Bit Bus</th><th>2-State Access</th><th>3-State Access</th><th>2-State Access</th><th>3-State Access</th></tr>
<tr><td>Instruction fetch</td><td>S_I</td><td rowspan="5">2</td><td rowspan="3">6</td><td rowspan="5">3</td><td rowspan="3">4</td><td rowspan="3">6 + 2 m</td><td rowspan="5">2</td><td rowspan="5">3 + m*</td></tr>
<tr><td>Branch address read</td><td>S_J</td></tr>
<tr><td>Stack operation</td><td>S_K</td></tr>
<tr><td>Byte data access</td><td>S_L</td><td>3</td><td>2</td><td>3 + m</td></tr>
<tr><td>Word data access</td><td>S_M</td><td>6</td><td>4</td><td>6 + 2 m</td></tr>
<tr><td>Internal operation</td><td>S_N</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Note: * For the MOVFPE and MOVTPE instructions, refer to the relevant microcontroller hardware manual.

Legend:

m: Number of wait states inserted into external device access

Table 2.8 Number of Cycles in Instruction Execution

Instruction	Mnemonic	Instruction Fetch I	Branch Address Read J	Stack Operation K	Byte Data Access L	Word Data Access M	Internal Operation N
ADD	ADD.B #xx:8,Rd	1					
	ADD.B Rs,Rd	1					
	ADD.W #xx:16,Rd	2					
	ADD.W Rs,Rd	1					
	ADD.L #xx:32,ERd	3					
	ADD.L ERs,ERd	1					
ADDS	ADDS #1/2/4,ERd	1					
ADDX	ADDX #xx:8,Rd	1					
	ADDX Rs,Rd	1					
AND	AND.B #xx:8,Rd	1					
	AND.B Rs,Rd	1					
	AND.W #xx:16,Rd	2					
	AND.W Rs,Rd	1					
	AND.L #xx:32,ERd	3					
	AND.L ERs,ERd	2					
ANDC	ANDC #xx:8,CCR	1					
BAND	BAND #xx:3,Rd	1					
	BAND #xx:3,@ERd	2			1		
	BAND #xx:3,@aa:8	2			1		
Bcc	BRA d:8 (BT d:8)	2					
	BRN d:8 (BF d:8)	2					
	BHI d:8	2					
	BLS d:8	2					
	BCC d:8 (BHS d:8)	2					
	BCS d:8 (BLO d:8)	2					
	BNE d:8	2					
	BEQ d:8	2					
	BVC d:8	2					
	BVS d:8	2					
	BPL d:8	2					
	BMI d:8	2					
	BGE d:8	2					
	BLT d:8	2					

Instruction	Mnemonic	Instruction Fetch I	Branch Address Read J	Stack Operation K	Byte Data Access L	Word Data Access M	Internal Operation N
Bcc	BGT d:8	2					
	BLE d:8	2					
	BRA d:16 (BT d:16)	2					2
	BRN d:16 (BF d:16)	2					2
	BHI d:16	2					2
	BLS d:16	2					2
	BCC d:16 (BHS d:16)	2					2
	BCS d:16 (BLO d:16)	2					2
	BNE d:16	2					2
	BEQ d:16	2					2
	BVC d:16	2					2
	BVS d:16	2					2
	BPL d:16	2					2
	BMI d:16	2					2
	BGE d:16	2					2
	BLT d:16	2					2
	BGT d:16	2					2
	BLE d:16	2					2
BCLR	BCLR #xx:3,Rd	1					
	BCLR #xx:3,@ERd	2			2		
	BCLR #xx:3,@aa:8	2			2		
	BCLR Rn,Rd	1					
	BCLR Rn,@ERd	2			2		
	BCLR Rn,@aa:8	2			2		
BIAND	BIAND #xx:3,Rd	1					
	BIAND #xx:3,@ERd	2			1		
	BIAND #xx:3,@aa:8	2			1		
BILD	BILD #xx:3,Rd	1					
	BILD #xx:3,@ERd	2			1		
	BILD #xx:3,@aa:8	2			1		
BIOR	BIOR #xx:8,Rd	1					
	BIOR #xx:8,@ERd	2			1		
	BIOR #xx:8,@aa:8	2			1		

			Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	**Mnemonic**		**I**	**J**	**K**	**L**	**M**	**N**
BIST	BIST #xx:3,Rd		1					
	BIST #xx:3,@ERd		2			2		
	BIST #xx:3,@aa:8		2			2		
BIXOR	BIXOR #xx:3,Rd		1					
	BIXOR #xx:3,@ERd		2			1		
	BIXOR #xx:3,@aa:8		2			1		
BLD	BLD #xx:3,Rd		1					
	BLD #xx:3,@ERd		2			1		
	BLD #xx:3,@aa:8		2			1		
BNOT	BNOT #xx:3,Rd		1					
	BNOT #xx:3,@ERd		2			2		
	BNOT #xx:3,@aa:8		2			2		
	BNOT Rn,Rd		1					
	BNOT Rn,@ERd		2			2		
	BNOT Rn,@aa:8		2			2		
BOR	BOR #xx:3,Rd		1					
	BOR #xx:3,@ERd		2			1		
	BOR #xx:3,@aa:8		2			1		
BSET	BSET #xx:3,Rd		1					
	BSET #xx:3,@ERd		2			2		
	BSET #xx:3,@aa:8		2			2		
	BSET Rn,Rd		1					
	BSET Rn,@ERd		2			2		
	BSET Rn,@aa:8		2			2		
BSR	BSR d:8	Advanced	2		2			
		Normal	2		1			
	BSR d:16	Advanced	2		2			2
		Normal	2		1			2
BST	BST #xx:3,Rd		1					
	BST #xx:3,@ERd		2			2		
	BST #xx:3,@aa:8		2			2		

Instruction	Mnemonic	Instruction Fetch I	Branch Address Read J	Stack Operation K	Byte Data Access L	Word Data Access M	Internal Operation N
BTST	BTST #xx:3,Rd	1					
	BTST #xx:3,@ERd	2			1		
	BTST #xx:3,@aa:8	2			1		
	BTST Rn,Rd	1					
	BTST Rn,@ERd	2			1		
	BTST Rn,@aa:8	2			1		
BXOR	BXOR #xx:3,Rd	1					
	BXOR #xx:3,@ERd	2			1		
	BXOR #xx:3,@aa:8	2			1		
CMP	CMP.B #xx:8,Rd	1					
	CMP.B Rs,Rd	1					
	CMP.W #xx:16,Rd	2					
	CMP.W Rs,Rd	1					
	CMP.L #xx:32,ERd	3					
	CMP.L ERs,ERd	1					
DAA	DAA Rd	1					
DAS	DAS Rd	1					
DEC	DEC.B Rd	1					
	DEC.W #1/2,Rd	1					
	DEC.L #1/2,ERd	1					
DIVXS	DIVXS.B Rs,Rd	2					12
	DIVXS.W Rs,ERd	2					20
DIVXU	DIVXU.B Rs,Rd	1					12
	DIVXU.W Rs,ERd	1					20
EEPMOV	EEPMOV.B	2			2n + 2*1		
	EEPMOV.W	2			2n + 2*1		
EXTS	EXTS.W Rd	1					
	EXTS.L ERd	1					
EXTU	EXTU.W Rd	1					
	EXTU.L ERd	1					
INC	INC.B Rd	1					
	INC.W #1/2,Rd	1					
	INC.L #1/2,ERd	1					

Instruction	Mnemonic		Instruction Fetch I	Branch Address Read J	Stack Operation K	Byte Data Access L	Word Data Access M	Internal Operation N
JMP	JMP @ERn		2					
	JMP @aa:24		2					2
	JMP @@aa:8	Advanced	2	2				2
		Normal	2	1				2
JSR	JSR @ERn	Advanced	2		2			
		Normal	2		1			
	JSR @aa:24	Advanced	2		2			2
		Normal	2		1			2
	JSR @@aa:8	Advanced	2	2	2			
		Normal	2	1	1			
LDC	LDC #xx:8,CCR		1					
	LDC Rs,CCR		1					
	LDC @ERs,CCR		2				1	
	LDC @(d:16,ERs),CCR		3				1	
	LDC @(d:24,ERs),CCR		5				1	
	LDC @ERs+,CCR		2				1	2
	LDC @aa:16,CCR		3				1	
	LDC @aa:24,CCR		4				1	
MOV	MOV.B #xx:8,Rd		1					
	MOV.B Rs,Rd		1					
	MOV.B @ERs,Rd		1			1		
	MOV.B @(d:16,ERs),Rd		2			1		
	MOV.B @(d:24,ERs),Rd		4			1		
	MOV.B @ERs+,Rd		1			1		2
	MOV.B @aa:8,Rd		1			1		
	MOV.B @aa:16,Rd		2			1		
	MOV.B @aa:24,Rd		3			1		
	MOV.B Rs,@ERd		1			1		
	MOV.B Rs,@(d:16,ERd)		2			1		
	MOV.B Rs,@(d:24,ERd)		4			1		
	MOV.B Rs,@–ERd		1			1		2
	MOV.B Rs,@aa:8		1			1		
	MOV.B Rs,@aa:16		2			1		
	MOV.B Rs,@aa:24		3			1		

Instruction	Mnemonic	Instruction Fetch I	Branch Address Read J	Stack Operation K	Byte Data Access L	Word Data Access M	Internal Operation N
MOV	MOV.W #xx:16,Rd	2					
	MOV.W Rs,Rd	1					
	MOV.W @ERs,Rd	1				1	
	MOV.W @(d:16,ERs),Rd	2				1	
	MOV.W @(d:24,ERs),Rd	4				1	
	MOV.W @ERs+,Rd	1				1	2
	MOV.W @aa:16,Rd	2				1	
	MOV.W @aa:24,Rd	3				1	
	MOV.W Rs,@ERd	1				1	
	MOV.W Rs,@(d:16,ERd)	2				1	
	MOV.W Rs,@(d:24,ERd)	4				1	
	MOV.W Rs,@–ERd	1				1	2
	MOV.W Rs,@aa:16	2				1	
	MOV.W Rs,@aa:24	3				1	
	MOV.L #xx:32,ERd	3					
	MOV.L ERs,ERd	1					
	MOV.L @ERs,ERd	2				2	
	MOV.L @(d:16,ERs),ERd	3				2	
	MOV.L @(d:24,ERs),ERd	5				2	
	MOV.L @ERs+,ERd	2				2	2
	MOV.L @aa:16,ERd	3				2	
	MOV.L @aa:24,ERd	4				2	
	MOV.L ERs,@ERd	2				2	
	MOV.L ERs,@(d:16,ERd)	3				2	
	MOV.L ERs,@(d:24,ERd)	5				2	
	MOV.L ERs,@–ERd	2				2	2
	MOV.L ERs,@aa:16	3				2	
	MOV.L ERs,@aa:24	4				2	
MOVFPE	MOVFPE @:aa:16,Rd	2			1*2		
MOVTPE	MOVTPE Rs,@:aa:16	2			1*2		
MULXS	MULXS.B Rs,Rd	2					12
	MULXS.W Rs,ERd	2					20
MULXU	MULXU.B Rs,Rd	1					12
	MULXU.W Rs,ERd	1					20

			Instruction Fetch	Branch Address Read	Stack Operation	Byte Data Access	Word Data Access	Internal Operation
Instruction	**Mnemonic**		**I**	**J**	**K**	**L**	**M**	**N**
NEG	NEG.B Rd		1					
	NEG.W Rd		1					
	NEG.L ERd		1					
NOP	NOP		1					
NOT	NOT.B Rd		1					
	NOT.W Rd		1					
	NOT.L ERd		1					
OR	OR.B #xx:8,Rd		1					
	OR.B Rs,Rd		1					
	OR.W #xx:16,Rd		2					
	OR.W Rs,Rd		1					
	OR.L #xx:32,ERd		3					
	OR.L ERs,ERd		2					
ORC	ORC #xx:8,CCR		1					
POP	POP.W Rn		1				1	2
	POP.L ERn		2				2	2
PUSH	PUSH.W Rn		1				1	2
	PUSH.L ERn		1				2	2
ROTL	ROTL.B Rd		1					
	ROTL.W Rd		1					
	ROTL.L ERd		1					
ROTR	ROTR.B Rd		1					
	ROTR.W Rd		1					
	ROTR.L ERd		1					
ROTXL	ROTXL.B Rd		1					
	ROTXL.W Rd		1					
	ROTXL.L ERd		1					
ROTXR	ROTXR.B Rd		1					
	ROTXR.W Rd		1					
	ROTXR.L ERd		1					
RTE	RTE		2		2			2
RTS	RTS	Advanced	2		2			2
		Normal	2		1			2

Instruction	Mnemonic		Instruction Fetch I	Branch Address Read J	Stack Operation K	Byte Data Access L	Word Data Access M	Internal Operation N
SHAL	SHAL.B Rd		1					
	SHAL.W Rd		1					
	SHAL.L ERd		1					
SHAR	SHAR.B Rd		1					
	SHAR.W Rd		1					
	SHAR.L ERd		1					
SHLL	SHLL.B Rd		1					
	SHLL.W Rd		1					
	SHLL.L ERd		1					
SHLR	SHLR.B Rd		1					
	SHLR.W Rd		1					
	SHLR.L ERd		1					
SLEEP	SLEEP		1					
STC	STC CCR,Rd		1					
	STC CCR,@ERd		2				1	
	STC CCR,@(d:16,ERd)		3				1	
	STC CCR,@(d:24,ERd)		5				1	
	STC CCR,@–ERd		2				1	2
	STC CCR,@aa:16		3				1	
	STC CCR,@aa:24		4				1	
SUB	SUB.B Rs,Rd		1					
	SUB.W #xx:16,Rd		2					
	SUB.W Rs,Rd		1					
	SUB.L #xx:32,ERd		3					
	SUB.L ERs,ERd		1					
SUBS	SUBS #1/2/4,ERd		1					
SUBX	SUBX #xx:8,Rd		1					
	SUBX Rs,Rd		1					
TRAPA	TRAPA #x:2	Advanced	2	2	2			4
		Normal	2	1	2			4

Instruction	Mnemonic	Instruction Fetch I	Branch Address Read J	Stack Operation K	Byte Data Access L	Word Data Access M	Internal Operation N
XOR	XOR.B #xx:8,Rd	1					
	XOR.B Rs,Rd	1					
	XOR.W #xx:16,Rd	2					
	XOR.W Rs,Rd	1					
	XOR.L #xx:32,ERd	3					
	XOR.L ERs,ERd	2					
XORC	XORC #xx:8,CCR	1					

Notes: 1. When n bytes of data are transferred.

2.7 Condition Code Modification

This section indicates the effect of each CPU instruction on the condition code. The notation used in the table is defined below.

m	31 for longword operands, 15 for word operands, 7 for byte operands
S_i	The i-th bit of the source operand
D_i	The i-th bit of the destination operand
R_i	The i-th bit of the result
D_n	The specified bit in the destination operand
—	Not affected
↕	Modified according to the result of the instruction (see definition)
0	Always cleared to 0
1	Always set to 1
*	Undetermined (no guaranteed value)
Z'	Z flag before instruction execution
C'	C flag before instruction execution

Table 2.9 Condition Code Modification

Instruction	H	N	Z	V	C	Definition
ADD	↕	↕	↕	↕	↕	H = S m – 4 · D m – 4 + D m – 4 · / R m – 4 + S m – 4 · / R m – 4 N = R m Z = / R m · / R m – 1 · ... · / R 0 V = S m · D m · / R m + / S m · / D m · R m C = S m · D m + D m · / R m + S m · / R m
ADDS	—	—	—	—	—	
ADDX	↕	↕	↕	↕	↕	H = S m – 4 · D m – 4 + D m – 4 · / R m – 4 + S m – 4 · / R m – 4 N = R m Z = Z ' · / R m · ... · / R 0 V = S m · D m · / R m + / S m · / D m · R m C = S m · D m + D m · / R m + S m · / R m
AND	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
ANDC	↕	↕	↕	↕	↕	Stores the corresponding bits of the result
BAND	—	—	—	—	↕	C = C ' · D n
Bcc	—	—	—	—	—	
BCLR	—	—	—	—	—	
BIAND	—	—	—	—	↕	C = C ' · / D n
BILD	—	—	—	—	↕	C = / D n
BIOR	—	—	—	—	↕	C = C ' + / D n
BIST	—	—	—	—	—	
BIXOR	—	—	—	—	↕	C = C ' · / D n + / C ' · / D n
BLD	—	—	—	—	↕	C = D n
BNOT	—	—	—	—	—	
BOR	—	—	—	—	↕	C = C ' + D n
BSET	—	—	—	—	—	
BSR	—	—	—	—	—	
BST	—	—	—	—	—	
BTST	—	—	↕	—	—	Z = / D n
BXOR	—	—	—	—	↕	C = C ' · / D n + / C ' · D n

Instruction	H	N	Z	V	C	Definition
CMP	↕	↕	↕	↕	↕	H = S m – 4 · / D m – 4 + / D m – 4 · R m – 4 + S m – 4 · R m – 4 N = R m Z = / R m · / R m – 1 · ... · / R 0 V = / S m · D m · / R m + S m · / D m · R m C = S m · / D m + / D m · R m + S m · R m
DAA	*	↕	↕	*	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C: decimal arithmetic carry
DAS	*	↕	↕	*	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C: decimal arithmetic borrow
DEC	—	↕	↕	↕	—	N = R m Z = / R m· / R m – 1 · ... · / R 0 V = D m · / R m
DIVXS	—	↕	↕	—	—	N = S m · / D m + / S m · D m Z = / S m · / S m – 1 · ... · / S 0
DIVXU	—	↕	↕	—	—	N = S m Z = / S m · / S m – 1 · ... · / S 0
EEPMOV	—	—	—	—	—	
EXTS	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
EXTU	—	O	↕	O	—	Z = / R m · / R m – 1 · ... · / R 0
INC	—	↕	↕	↕	—	N = R m Z = / R m · / R m – 1 · ... · / R 0 V = D m · / R m
JMP	—	—	—	—	—	
JSR	—	—	—	—	—	
LDC	↕	↕	↕	↕	↕	Stores the corresponding bits of the result
MOV	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
MOVFPE	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
MOVTPE	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0

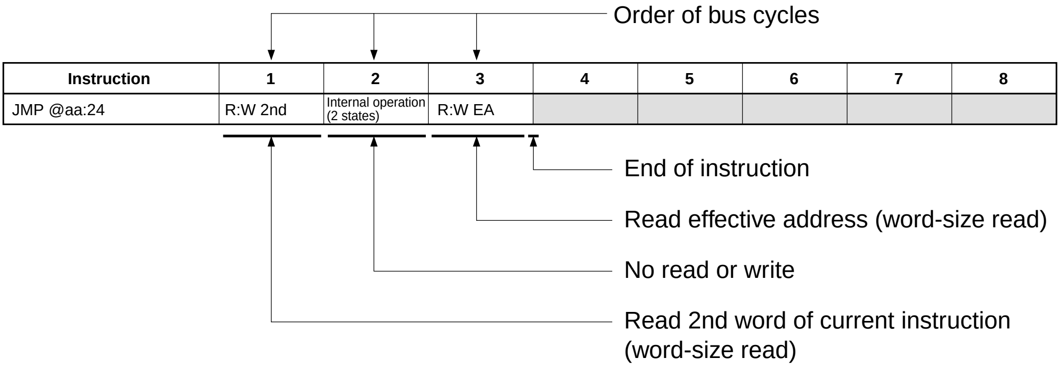
Instruction	H	N	Z	V	C	Definition
MULXS	—	↕	↕	—	—	N = R 2 m Z = R 2 m · R 2 m – 1 · ... · / R 0
MULXU	—	—	—	—	—	
NEG	↕	↕	↕	↕	↕	H = D m – 4 + R m – 4 N = R m Z = / R m · / R m – 1 · ... · R 0 V = D m · R m C = D m + R m
NOP	—	—	—	—	—	
NOT	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
OR	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · .... · / R 0
ORC	↕	↕	↕	↕	↕	Stores the corresponding bits of the result
POP	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
PUSH	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
ROTL	—	↕	↕	O	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C = D m
ROTR	—	↕	↕	O	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C = D 0
ROTXL	—	↕	↕	O	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C = D m
ROTXR	—	↕	↕	O	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C = D 0
RTS	—	—	—	—	—	
RTE	↕	↕	↕	↕	↕	Stores the corresponding bits of the result

Instruction	H	N	Z	V	C	Definition
SHAL	—	↕	↕	↕	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 V = D m · / D m – 1 + / D m · D m – 1 C = D m
SHAR	—	↕	↕	O	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C = D 0
SHLL	—	↕	↕	O	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C = D m
SHLR	—	↕	↕	O	↕	N = R m Z = / R m · / R m – 1 · ... · / R 0 C = D 0
SLEEP	—	—	—	—	—	
STC	—	—	—	—	—	
SUB	↕	↕	↕	↕	↕	H = S m – 4 · / D m – 4 + / D m – 4 · R m – 4 + S m – 4 · R m – 4 N = R m Z = / R m · / R m – 1 · ... · / R 0 V = / S m · D m · / R m + S m · / D m · R m C = S m · / D m + / D m · R m + S m · R m
SUBS	—	—	—	—	—	
SUBX	↕	↕	↕	↕	↕	H = S m – 4 · / D m – 4 + / D m – 4 · R m – 4 + S m – 4 · R m – 4 N = R m Z = Z ' · / R m · ... · / R 0 V = / S m · D m · / R m + S m · / D m · R m C = S m · / D m + / D m · R m + S m · R m
TRAPA	—	—	—	—	—	
XOR	—	↕	↕	O	—	N = R m Z = / R m · / R m – 1 · ... · / R 0
XORC	↕	↕	↕	↕	↕	Stores the corresponding bits of the result

2.8 Bus Cycles During Instruction Execution

Table 2.10 indicates the bus cycles during instruction execution by the H8/300H CPU. For the number of states per bus cycle, see table 2.7, Number of States per Cycle.

How to read the table:



Legend

R:B	Byte-size read
R:W	Word-size read
W:B	Byte-size write
W:W	Word-size write
2nd	Address of 2nd word (3rd and 4th bytes)
3rd	Address of 3rd word (5th and 6th bytes)
4th	Address of 4th word (7th and 8th bytes)
5th	Address of 5th word (9th and 10th bytes)
NEXT	Address of next instruction
EA	Effective address
VEC	Vector address

Figure 2.1 shows timing waveforms for the address bus and the $\overline{RD}$ and $\overline{WR}$ ($\overline{HWR}$ or $\overline{LWR}$) signals during execution of the above instruction with an 8-bit bus, using 3-state access with no wait states.

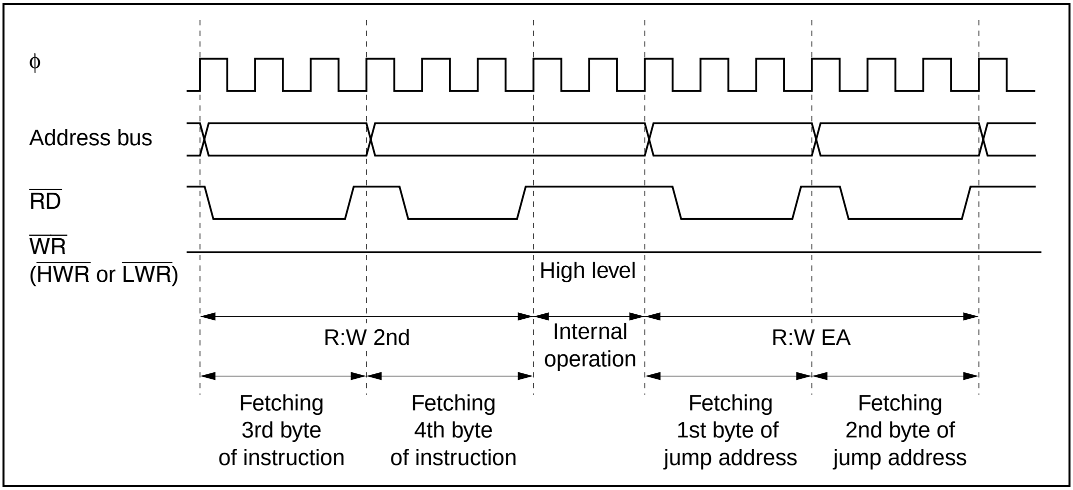


Figure 2.1 Address Bus, $\overline{RD}$, and $\overline{WR}$ ($\overline{HWR}$ or $\overline{LWR}$) Timing (8-bit bus, 3-state access, no wait states)

Table 2.10 Bus States

Instruction	1	2	3	4	5	6	7	8
ADD.B #xx:8,Rd	R:W NEXT							
ADD.B Rs,Rd	R:W NEXT							
ADD.W #xx:16,Rd	R:W 2nd	R:W NEXT						
ADD.W Rs,Rd	R:W NEXT							
ADD.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT					
ADD.L ERs,ERd	R:W NEXT							
ADDS #1/2/4,ERd	R:W NEXT							
ADDX #xx:8,Rd	R:W NEXT							
ADDX Rs,Rd	R:W NEXT							
AND.B #xx:8,Rd	R:W NEXT							
AND.B Rs,Rd	R:W NEXT							
AND.W #xx:16,Rd	R:W 2nd	R:W NEXT						
AND.W Rs,Rd	R:W NEXT							
AND.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT					
AND.L ERs,ERd	R:W 2nd	R:W NEXT						
ANDC #xx:8,CCR	R:W NEXT							
BAND #xx:3,Rd	R:W NEXT							
BAND #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT					
BAND #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT					
BRA d:8 (BT d;8)	R:W NEXT	R:W EA						
BRN d:8 (BF d;8)	R:W NEXT	R:W EA						
BHI d:8	R:W NEXT	R:W EA						
BLS d:8	R:W NEXT	R:W EA						
BCC d:8 (BHS d;8)	R:W NEXT	R:W EA						
BCS d:8 (BLO d;8)	R:W NEXT	R:W EA						
BNE d:8	R:W NEXT	R:W EA						
BEQ d:8	R:W NEXT	R:W EA						
BVC d:8	R:W NEXT	R:W EA						
BVS d:8	R:W NEXT	R:W EA						
BPL d:8	R:W NEXT	R:W EA						
BMI d:8	R:W NEXT	R:W EA						

Instruction	1	2	3	4	5	6	7	8
BGE d:8	R:W NEXT	R:W EA						
BLT d:8	R:W NEXT	R:W EA						
BGT d:8	R:W NEXT	R:W EA						
BLE d:8	R:W NEXT	R:W EA						
BRA d:16 (BT d;16)	R:W 2nd	Internal operation, 2 states	R:W EA					
BRN d:16 (BF d;16)	R:W 2nd	Internal operation, 2 states	R:W EA					
BHI d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BLS d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BCC d:16 (BHS d;16)	R:W 2nd	Internal operation, 2 states	R:W EA					
BCS d:16 (BLO d;16)	R:W 2nd	Internal operation, 2 states	R:W EA					
BNE d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BEQ d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BVC d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BVS d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BPL d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BMI d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BGE d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BLT d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BGT d:16	R:W 2nd	Internal operation, 2 states	R:W EA					
BLE d:16	R:W 2nd	Internal operation, 2 states	R:W EA					

Instruction	1	2	3	4	5	6	7	8
BCLR #xx:3,Rd	R:W NEXT							
BCLR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BCLR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BCLR Rn,Rd	R:W NEXT							
BCLR Rn,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BCLR Rn,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BIAND #xx:3,Rd	R:W NEXT							
BIAND #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT					
BIAND #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT					
BILD #xx:3,Rd	R:W NEXT							
BILD #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT					
BILD #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT					
BIOR #xx:8,Rd	R:W NEXT							
BIOR #xx:8,@ERd	R:W 2nd	R:B EA	R:W NEXT					
BIOR #xx:8,@aa:8	R:W 2nd	R:B EA	R:W NEXT					
BIST #xx:3,Rd	R:W NEXT							
BIST #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BIST #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BIXOR #xx:3,Rd	R:W NEXT							
BIXOR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT					
BIXOR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT					
BLD #xx:3,Rd	R:W NEXT							
BLD #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT					
BLD #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT					
BNOT #xx:3,Rd	R:W NEXT							
BNOT #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BNOT #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BNOT Rn,Rd	R:W NEXT							
BNOT Rn,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BNOT Rn,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BOR #xx:3,Rd	R:W NEXT							
BOR #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT					
BOR #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT					
BSET #xx:3,Rd	R:W NEXT							
BSET #xx:3,@ERd	R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BSET #xx:3,@aa:8	R:W 2nd	R:B EA	R:W NEXT	W:B EA				

Instruction		1	2	3	4	5	6	7	8
BSET Rn,Rd		R:W NEXT							
BSET Rn,@ERd		R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BSET Rn,@aa:8		R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BRS d:8	Normal	R:W NEXT	R:W EA	W:W Stack					
	Advanced	R:W NEXT	R:W EA	W:W Stack (H)	W:W Stack (L)				
BRS d:16	Normal	R:W 2nd	Internal operation, 2 states	R:W EA	W:W Stack				
	Advanced	R:W 2nd	Internal operation, 2 states	R:W EA	W:W Stack (H)	W:W Stack (L)			
BST #xx:3,Rd		R:W NEXT							
BST #xx:3,@ERd		R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BST #xx:3,@aa:8		R:W 2nd	R:B EA	R:W NEXT	W:B EA				
BTST #xx:3,Rd		R:W NEXT							
BTST #xx:3,@ERd		R:W 2nd	R:B EA	R:W NEXT					
BTST #xx:3,@aa:8		R:W 2nd	R:B EA	R:W NEXT					
BTST Rn,Rd		R:W NEXT							
BTST Rn,@ERd		R:W 2nd	R:B EA	R:W NEXT					
BTST Rn,@aa:8		R:W 2nd	R:B EA	R:W NEXT					
BXOR #xx:3,Rd		R:W NEXT							
BXOR #xx:3,@ERd		R:W 2nd	R:B EA	R:W NEXT					
BXOR #xx:3,@aa:8		R:W 2nd	R:B EA	R:W NEXT					
CMP.B #xx:8,Rd		R:W NEXT							
CMP.B Rs,Rd		R:W NEXT							
CMP.W #xx:16,Rd		R:W 2nd	R:W NEXT						
CMP.W Rs,Rd		R:W NEXT							
CMP.L #xx:32,ERd		R:W 2nd	R:W 3rd	R:W NEXT					
CMP.L ERs,ERd		R:W NEXT							
DAA Rd		R:W NEXT							
DAS Rd		R:W NEXT							
DEC.B Rd		R:W NEXT							
DEC.W #1/2,Rd		R:W NEXT							
DEC.L #1/2,ERd		R:W NEXT							
DIVXS.B Rs,Rd		R:W 2nd	R:W NEXT	Internal operation, 12 states					
DIVXS.W Rs,ERd		R:W 2nd	R:W NEXT	Internal operation, 20 states					
DIVXU.B Rs,Rd		R:W NEXT	Internal operation, 12 states						
DIVXU.W Rs,ERd		R:W NEXT	Internal operation, 20 states						
EEPMOV.B		R:W 2nd	R:B EAs *1	R:B EAd *1	R:B EAs *2	W:B EAd *2	R:W NEXT		
EEPMOV.W		R:W 2nd	R:B EAs *1	R:B EAd *1	R:B EAs *2	W:B EAd *2	R:W NEXT		

Instruction		1	2	3	4	5	6	7	8
EXTS.W Rd		R:W NEXT							
EXTS.L ERd		R:W NEXT							
EXTU.W Rd		R:W NEXT							
EXTU.L ERd		R:W NEXT							
INC.B Rd		R:W NEXT							
INC.W #1/2,Rd		R:W NEXT							
INC.L #1/2,ERd		R:W NEXT							
JMP @ERn		R:W NEXT	R:W EA						
JMP @aa:24		R:W 2nd	Internal operation, 2 states	R:W EA					
JMP @@aa:8	Normal	R:W NEXT	R:W aa:8	Internal operation, 2 states	R:W EA				
	Advanced	R:W NEXT	R:W aa:8	R:W aa:8	Internal operation, 2 states	R:W EA			
JSR @ERn	Normal	R:W NEXT	R:W EA	W:W Stack					
	Advanced	R:W NEXT	R:W EA	W:W Stack (H)	W:W Stack (L)				
JSR @aa:24	Normal	R:W 2nd	Internal operation, 2 states	R:W EA	W:W Stack				
	Advanced	R:W 2nd	Internal operation, 2 states	R:W EA	W:W Stack (H)	W:W Stack (L)			
JSR @@aa:8	Normal	R:W NEXT	R:W aa:8	W:W Stack	R:W EA				
	Advanced	R:W NEXT	R:W aa:8	R:W aa:8	W:W Stack (H)	W:W Stack (L)	R:W EA		
LDC #xx:8,CCR		R:W NEXT							
LDC Rs,CCR		R:W NEXT							
LDC @ERs,CCR		R:W 2nd	R:W NEXT	R:W EA					
LDC @(d:16,ERs),CCR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA				
LDC @(d:24,ERs),CCR		R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	R:W EA		
LDC @ERs+,CCR		R:W 2nd	R:W NEXT	Internal operation, 2 states	R:W EA				
LDC @aa:16,CCR		R:W 2nd	R:W 3rd	R:W NEXT	R:W EA				
LDC @aa:24,CCR		R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA			
MOV.B #xx:8,Rd		R:W NEXT							
MOV.B Rs,Rd		R:W NEXT							
MOV.B @ERs,Rd		R:W NEXT	R:B EA						
MOV.B @(d:16,ERs),Rd		R:W 2nd	R:W NEXT	R:B EA					
MOV.B @(d:24,ERs),Rd		R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:B EA			

Instruction	1	2	3	4	5	6	7	8
MOV.B @ERs+,Rd	R:W NEXT	Internal operation, 2 states	R:B EA					
MOV.B @aa:8,Rd	R:W NEXT	R:B EA						
MOV.B @aa:16,Rd	R:W 2nd	R:W NEXT	R:B EA					
MOV.B @aa:24,Rd	R:W 2nd	R:W 3rd	R:W NEXT	R:B EA				
MOV.B Rs,@ERd	R:W NEXT	W:B EA						
MOV.B Rs,@(d:16,ERd)	R:W 2nd	R:W NEXT	W:B EA					
MOV.B Rs,@(d:24,ERd)	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:B EA			
MOV.B Rs,@–ERd	R:W NEXT	Internal operation, 2 states	W:B EA					
MOV.B Rs,@aa:8	R:W NEXT	W:B EA						
MOV.B Rs,@aa:16	R:W 2nd	R:W NEXT	W:B EA					
MOV.B Rs,@aa:24	R:W 2nd	R:W 3rd	R:W NEXT	W:B EA				
MOV.W #xx:16,Rd	R:W 2nd	R:W NEXT						
MOV.W Rs,Rd	R:W NEXT							
MOV.W @ERs,Rd	R:W NEXT	R:W EA						
MOV.W @(d:16,ERs),Rd	R:W 2nd	R:W NEXT	R:W EA					
MOV.W @(d:24,ERs),Rd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA			
MOV.W @ERs+,Rd	R:W NEXT	Internal operation, 2 states	R:W EA					
MOV.W @aa:16,Rd	R:W 2nd	R:W NEXT	R:W EA					
MOV.W @aa:24,Rd	R:W 2nd	R:W 3rd	R:W NEXT	R:B EA				
MOV.W Rs,@ERd	R:W NEXT	W:W EA						
MOV.W Rs,@(d:16,ERd)	R:W 2nd	R:W NEXT	W:W EA					
MOV.W Rs,@(d:24,ERd)	R:W 2nd	R:W 3rd	R:E 4th	R:W NEXT	W:W EA			
MOV.W Rs,@–ERd	R:W NEXT	Internal operation, 2 states	W:W EA					
MOV.W Rs,@aa:16	R:W 2nd	R:W NEXT	W:W EA					
MOV.W Rs,@aa:24	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA				
MOV.L #xx:32,ERd	R:W 2nd	R:W 3rd	R:W NEXT					
MOV.L ERs,ERd	R:W NEXT							
MOV.L @ERs,ERd	R:W 2nd	R:W NEXT	R:W EA	R:W EA+2				
MOV.L @(d:16,ERs),ERd	R:W 2nd	R:W 3rd	R:W NEXT	R:W EA	R:W EA+2			
MOV.L @(d:24,ERs),ERd	R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	R:W EA	R:W EA+2	
MOV.L @ERs+,ERd	R:W 2nd	R:W NEXT	Internal operation, 2 states	R:W EA	R:W EA+2			

Instruction	1	2	3	4	5	6	7	8
MOV.L @aa:16,ERd	R:W 2nd	R:W 3rd	R:W NEXT	R:W EA	R:W EA+2			
MOV.L @aa:24,ERd	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	R:W EA	R:W EA+2		
MOV.L ERs,@ERd	R:W 2nd	R:W NEXT	W:W EA	W:W EA+2				
MOV.L ERs,@(d:16,ERd)	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA				
MOV.L ERs,@(d:24,ERd)	R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	W:W EA	W:W EA+2	
MOV.L ERs,@–ERd	R:W 2nd	R:W NEXT	Internal operation, 2 states	W:W EA	W:W EA+2			
MOV.L ERs,@aa:16	R:W 2nd	R:W 3rd	R:W NEXT	W:W EA	W:W EA+2			
MOV.L ERs,@aa:24	R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA	W:W EA+2		
MOVFPE @aa:16,Rd	R:W 2nd	Internal operation, 2 states	R:W *3 EA					
MOVTPE Rs,@aa:16	R:W 2nd	Internal operation, 2 states	W:B *3 EA					
MULXS.B Rs,Rd	R:W 2nd	R:W NEXT	Internal operation, 12 states					
MULXS.W Rs,ERd	R:W 2nd	R:W NEXT	Internal operation, 20 states					
MULXU.B Rs,Rd	R:W NEXT	Internal operation, 12 states						
MULXU.W Rs,ERd	R:W NEXT	Internal operation, 20 states						
NEG.B Rd	R:W NEXT							
NEG.W Rd	R:W NEXT							
NEG.L ERd	R:W NEXT							
NOP	R:W NEXT							
NOT.B Rd	R:W NEXT							
NOT.W Rd	R:W NEXT							
NOT.L ERd	R:W NEXT							
OR.B #xx:8,Rd	R:W NEXT							
OR.B Rs,Rd	R:W NEXT							
OR.W #xx:16,Rd	R:W 2nd	R:W NEXT						
OR.W Rs,Rd	R:W NEXT							
OR.L #xx:32,ERd	R:W 2nd	R:W rd	R:W NEXT					
OR.L ERs,ERd	R:W 2nd	R:W NEXT						
ORC #xx:8,CCR	R:W NEXT							
POP.W Rn	R:W NEXT	Internal operation, 2 states	R:W Stack					
POP.L ERn	R:W 2nd	R:W NEXT	Internal operation, 2 states	R:W Stack (H)	R:W Stack (L)			

Instruction		1	2	3	4	5	6	7	8
PUSH.W Rn		R:W NEXT	Internal operation, 2 states	W:W Stack					
PUSH.L ERn		R:W 2nd	R:W NEXT	Internal operation, 2 states	W:W Stack (L)	W:W Stack (H)			
ROTL.B Rd		R:W NEXT							
ROTL.W Rd		R:W NEXT							
ROTL.L ERd		R:W NEXT							
ROTR.B Rd		R:W NEXT							
ROTR.W Rd		R:W NEXT							
ROTR.L ERd		R:W NEXT							
ROTXL.B Rd		R:W NEXT							
ROTXL.W Rd		R:W NEXT							
ROTXL.L ERd		R:W NEXT							
ROTXR.B Rd		R:W NEXT							
ROTXR.W Rd		R:W NEXT							
ROTXR.L ERd		R:W NEXT							
RTE		R:W NEXT	R:W Stack (H)	R:W Stack (L)	Internal operation, 2 states	R:W (*4)			
RTS	Normal	R:W NEXT	R:W Stack	Internal operation, 2 states	R:W (*4)				
	Advanced	R:W NEXT	R:W Stack (H)	R:W Stack (L)	Internal operation, 2 states	R:W (*4)			
SHAL.B Rd		R:W NEXT							
SHAL.W Rd		R:W NEXT							
SHAL.L ERd		R:W NEXT							
SHAR.B Rd		R:W NEXT							
SHAR.W Rd		R:W NEXT							
SHAR.L ERd		R:W NEXT							
SHLL.B Rd		R:W NEXT							
SHLL.W Rd		R:W NEXT							
SHLL.L ERd		R:W NEXT							
SHLR.B Rd		R:W NEXT							
SHLR.W Rd		R:W NEXT							
SHLR.L ERd		R:W NEXT							
SLEEP		R:W NEXT							
STC CCR,Rd		R:W NEXT							

Instruction		1	2	3	4	5	6	7	8
STC CCR,@ERd		R:W 2nd	R:W NEXT	W:W EA					
STC CCR,@(d:16,ERd)		R:W 2nd	R:W 3rd	R:W NEXT	W:W EA				
STC CCR,@(d:24,ERd)		R:W 2nd	R:W 3rd	R:W 4th	R:W 5th	R:W NEXT	W:W EA		
STC CCR,@–ERd		R:W 2nd	R:W NEXT	Internal operation, 2 states	W:W EA				
STC CCR,@aa:16		R:W 2nd	R:W 3rd	R:W NEXT	W:W EA				
STC CCR,@aa:24		R:W 2nd	R:W 3rd	R:W 4th	R:W NEXT	W:W EA			
SUB.B Rs,Rd		R:W NEXT							
SUB.W #xx:16,Rd		R:W 2nd	R:W NEXT						
SUB.W Rs,Rd		R:W NEXT							
SUB.L #xx:32,ERd		R:W 2nd	R:W 3rd	R:W NEXT					
SUB.L ERs,ERd		R:W NEXT							
SUBS #1/2/4,ERd		R:W NEXT							
SUBX #xx:8,Rd		R:W NEXT							
SUBX Rs,Rd		R:W NEXT							
TRAPA #x:2	Normal	R:W NEXT	Internal operation, 2 states	W:W Stack (L)	W:W Stack (H)	R:W VEC	Internal operation, 2 states	R:W (*7)	
	Advanced	R:W NEXT	Internal operation, 2 states	W:W Stack (L)	W:W Stack (H)	R:W VEC	R:W VEC+2	Internal operation, 2 states	R:W (*7)
XOR.B #xx8,Rd		R:W NEXT							
XOR.B Rs,Rd		R:W NEXT							
XOR.W #xx:16,Rd		R:W 2nd	R:W NEXT						
XOR.W Rs,Rd		R:W NEXT							
XOR.L #xx:32,ERd		R:W 2nd	R:W 3rd	R:W NEXT					
XOR.L ERs,ERd		R:W 2nd	R:W NEXT						
XORC #xx:8,CCR		R:W NEXT							
Reset exception handling	Normal	R:W VEC	Internal operation, 2 states	R:W (*5)					
	Advanced	R:W VEC	R:W VEC+2	Internal operation, 2 states	R:W (*5)				
Interrupt exception handling	Normal	R:W (*6)	Internal operation, 2 states	W:W stack (L)	W:W stack (H)	R:W VEC	Internal operation, 2 states	R:W (*7)	
	Advanced	R:W (*6)	Internal operation, 2 states	W:W stack (L)	W:W stack (H)	R:W VEC	R:W VEC+2	Internal operation, 2 states	R:W (*7)

Notes: 1. EAs is the contents of ER5. EAd is the contents of R6.

2. EAs is the contents of ER5. EAd is the contents of R6. Both registers are incremented by 1 after execution of the instruction. n is the initial value of R4L or R4. If n = 0, these bus cycles are not executed.

3. The number of states required for byte read or write varies from 9 to 16.

4. Starting address after return.

5. Starting address of the program.

6. Prefetch address, equal to two plus the PC value pushed on the stack. In recovery from sleep mode or software standby mode the read operation is replaced by an internal operation.

7. Starting address of the interrupt-handling routine.

8. NEXT: Next address after the current instruction.

   2nd: Address of the second word of the current instruction.

   3rd: Address of the third word of the current instruction.

   4th: Address of the fourth word of the current instruction.

   5th: Address of the fifth word of the current instruction.

   EA: Effective address.

   VEC: Vector address.

Section 3 Processing States

3.1 Overview

The CPU has five main processing states: the program execution state, exception handling state, power-down state, reset state, and bus-released state. The power-down state includes sleep mode, software standby mode, and hardware standby mode. Figure 3.1 shows a diagram of the processing states. Figure 3.2 indicates the state transitions. For details, refer to the relevant microcontroller hardware manual.

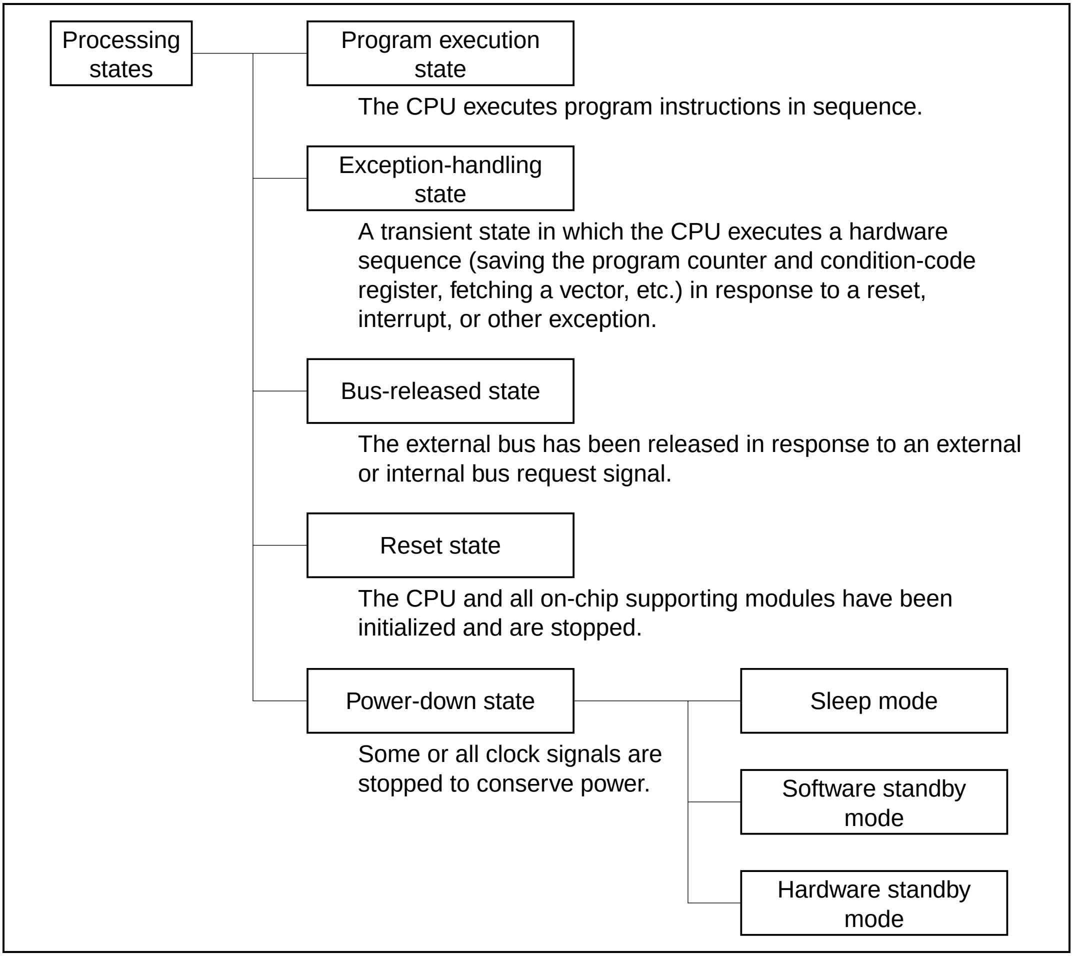
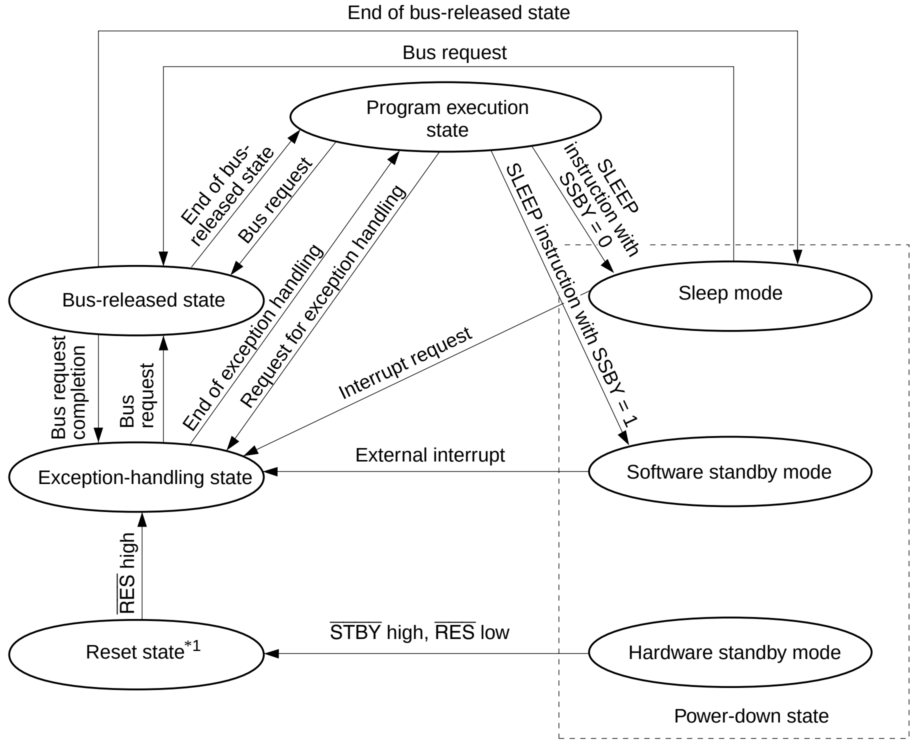


Figure 3.1 Processing States



Notes: 1. From any state except hardware standby mode, a transition to the reset state occurs whenever $\overline{\text{RES}}$ goes low.
2. From any state, a transition to hardware standby mode occurs when $\overline{\text{STBY}}$ goes low.

Figure 3.2 State Transitions

3.2 Program Execution State

In this state the CPU executes program instructions in normal sequence.

3.3 Exception-Handling State

The exception-handling state is a transient state that occurs when the CPU alters the normal program flow due to a reset, interrupt, or trap instruction. The CPU fetches a starting address from the exception vector table and branches to that address. In interrupt exception handling the CPU references the stack pointer (ER7) and saves the program counter and condition-code register.

3.3.1 Types of Exception Handling and Their Priority

Exception handling is performed for resets, interrupts, and trap instructions. Table 3.1 indicates the types of exception handling and their priority.

Table 3.1 Exception Handling Types and Priority

Priority	Type of Exception	Detection Timing	Start of Exception Handling
High ↑	Reset	Synchronized with clock	Exception handling starts immediately when $\overline{RES}$ changes from low to high
	Interrupt	End of instruction execution (see note)	When an interrupt is requested, exception handling starts at the end of the current instruction or current exception-handling sequence
Low	Trap instruction	When TRAPA instruction is executed	Exception handling starts when a trap (TRAPA) instruction is executed

Note: Interrupts are not detected at the end of the ANDC, ORC, XORC, and LDC instructions, or immediately after reset exception handling.

Figure 3.3 classifies the exception sources. For further details about exception sources, vector numbers, and vector addresses refer to the relevant microcontroller hardware manual.

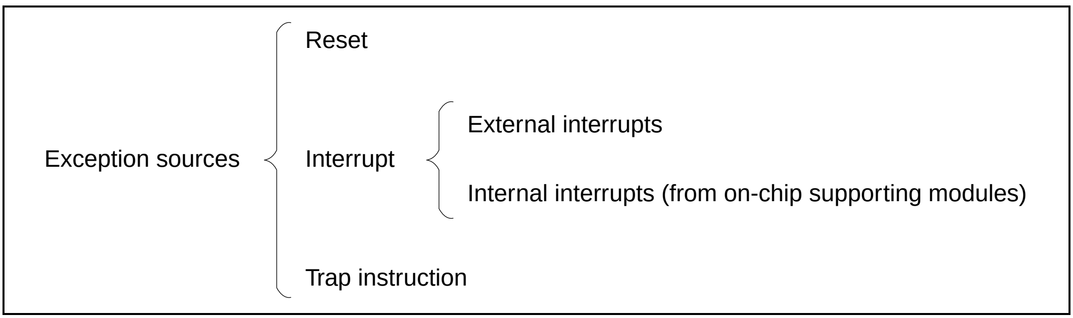


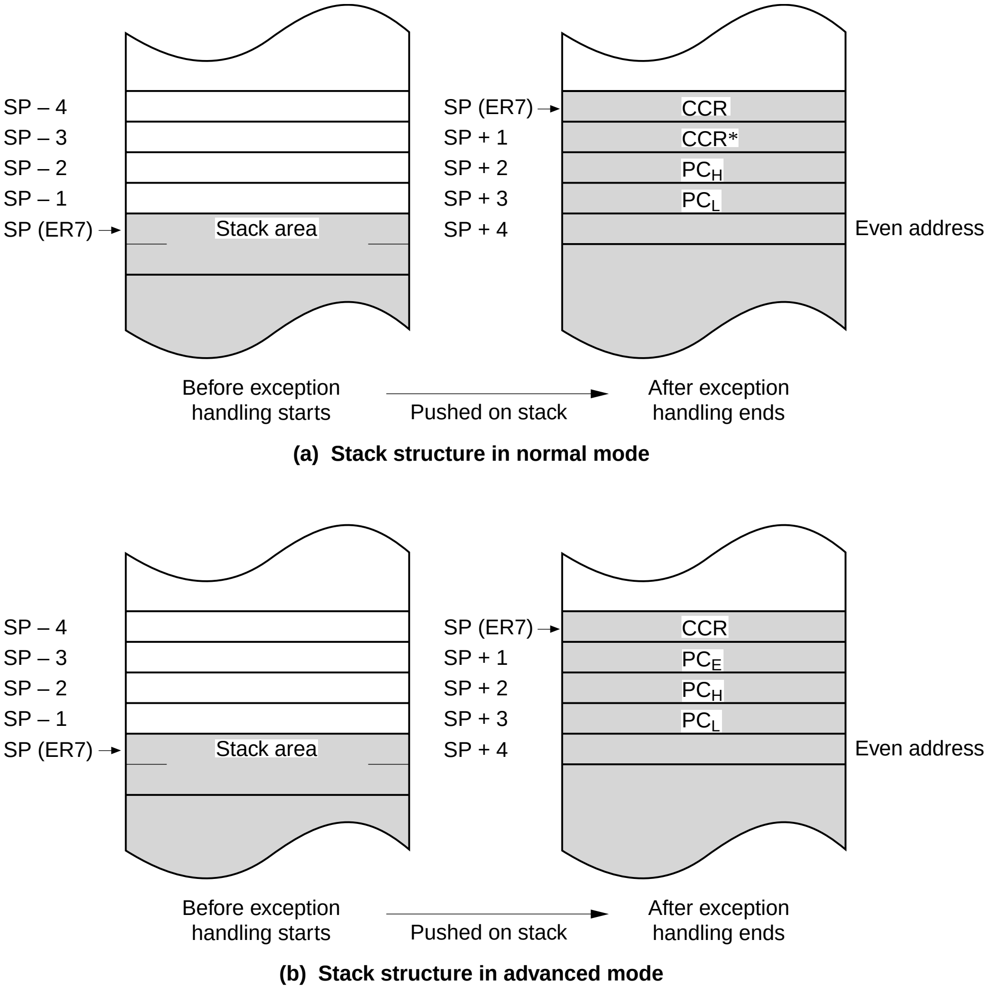
Figure 3.3 Classification of Exception Sources

3.3.2 Exception-Handling Sequences

Reset Exception Handling: Reset exception handling has the highest priority. The reset state is entered when the $\overline{\text{RES}}$ signal goes low. Then, if $\overline{\text{RES}}$ goes high again, reset exception handling starts when the reset condition is satisfied. Refer to the relevant microcontroller hardware manual for details about the reset condition. When reset exception handling starts the CPU fetches a start address from the exception vector table and starts program execution from that address. All interrupts, including NMI, are disabled during the reset exception-handling sequence and immediately after it ends.

Interrupt Exception Handling and Trap Instruction Exception Handling: When these exception-handling sequences begin, the CPU references the stack pointer (ER7) and pushes the program counter and condition-code register on the stack. Next, if the UE bit in the system control register (SYSCR) is set to 1, the CPU sets the I bit in the condition-code register to 1. If the UE bit is cleared to 0, the CPU sets both the I bit and the UI bit in the condition-code register to 1. Then the CPU fetches a start address from the exception vector table and execution branches to that address.

The program-counter value pushed on the stack and the start address fetched from the vector table are 16 bits long in normal mode and 24 bits long in advanced mode. Figure 3.4 shows the stack after the exception-handling sequence.



Legend:

PC_E: Program counter (PC) bits 23 to 16
PC_H: Program counter (PC) bits 15 to 8
PC_L: Program counter (PC) bits 7 to 0
CCR: Condition code register
SP: Stack pointer

Notes: * Ignored at return.

1. PC is the address of the first instruction executed after the return from the exception-handling routine.
2. Registers must be saved and restored by word access or longword access, starting at an even address.

Figure 3.4 Stack Structure after Exception Handling

3.4 Bus-Released State

This is a state in which the bus has been released in response to a bus request from a bus master other than the CPU. While the bus is released, the CPU halts except for internal operations. For further details, refer to the relevant microcontroller hardware manual.

For further details, refer to the relevant microcontroller hardware manual.

3.5 Reset State

When the $\overline{\text{RES}}$ input goes low all current processing stops and the CPU enters the reset state. The I bit in the condition-code register is set to 1 by a reset. All interrupts are masked in the reset state. Reset exception handling starts when the $\overline{\text{RES}}$ signal changes from low to high.

3.6 Power-Down State

In the power-down state the CPU stops operating to conserve power. There are three modes: sleep mode, software standby mode, and hardware standby mode. For details, refer to the relevant microcontroller hardware manual.

3.6.1 Sleep Mode

A transition to sleep mode is made if the SLEEP instruction is executed while the software standby bit (SSBY) is cleared to 0.

CPU operations stop immediately after execution of the SLEEP instruction. The contents of CPU registers are retained.

3.6.2 Software Standby Mode

A transition to software standby mode is made if the SLEEP instruction is executed while the SSBY bit is set to 1.

The CPU and clock halt and all on-chip supporting modules stop operating. The on-chip supporting modules are reset, but as long as a specified voltage is supplied the contents of CPU registers and on-chip RAM are retained. The I/O ports also remain in their existing states.

3.6.3 Hardware Standby Mode

A transition to hardware standby mode is made when the $\overline{\text{STBY}}$ input goes low.

As in software standby mode, the CPU and clock halt and the on-chip supporting modules are reset, but as long as a specified voltage is supplied, on-chip RAM contents are retained.

Section 4 Basic Timing

4.1 Overview

The CPU is driven by a clock, denoted by the symbol ϕ. One cycle of the clock is referred to as a “state.” The memory cycle or bus cycle consists of two or three states. Different methods are used to access on-chip memory, on-chip supporting modules, and external devices. Refer to the relevant microcontroller hardware manual for details.

4.2 On-Chip Memory (RAM, ROM)

For high-speed processing, on-chip memory is accessed in two states. The data bus is 16 bits wide, permitting both byte and word access. Figure 4.1 shows the on-chip memory access cycle. Figure 4.2 shows the pin states.

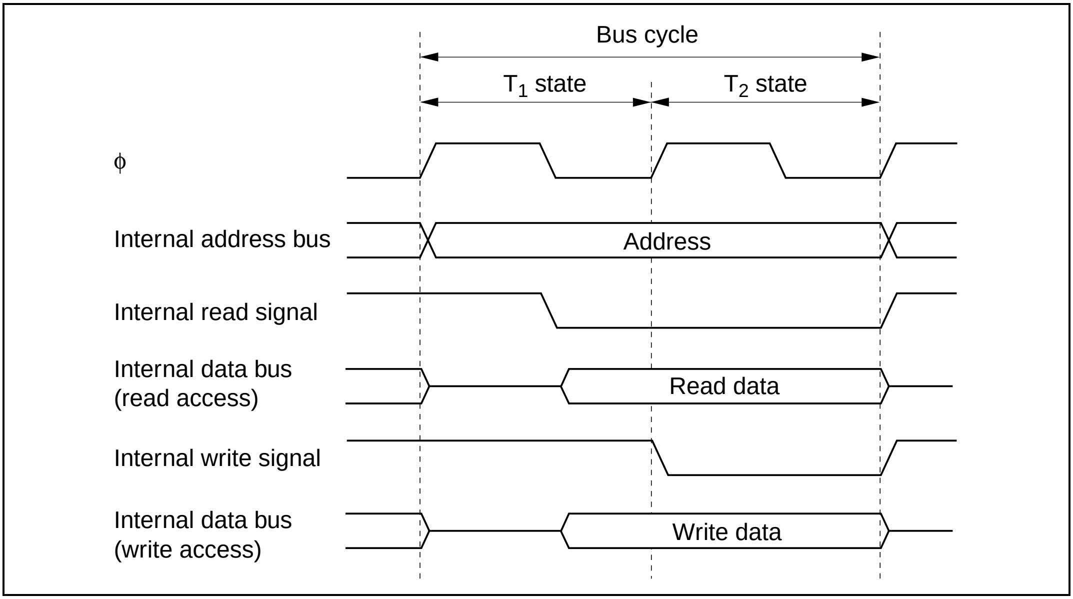


Figure 4.1 On-Chip Memory Access Cycle

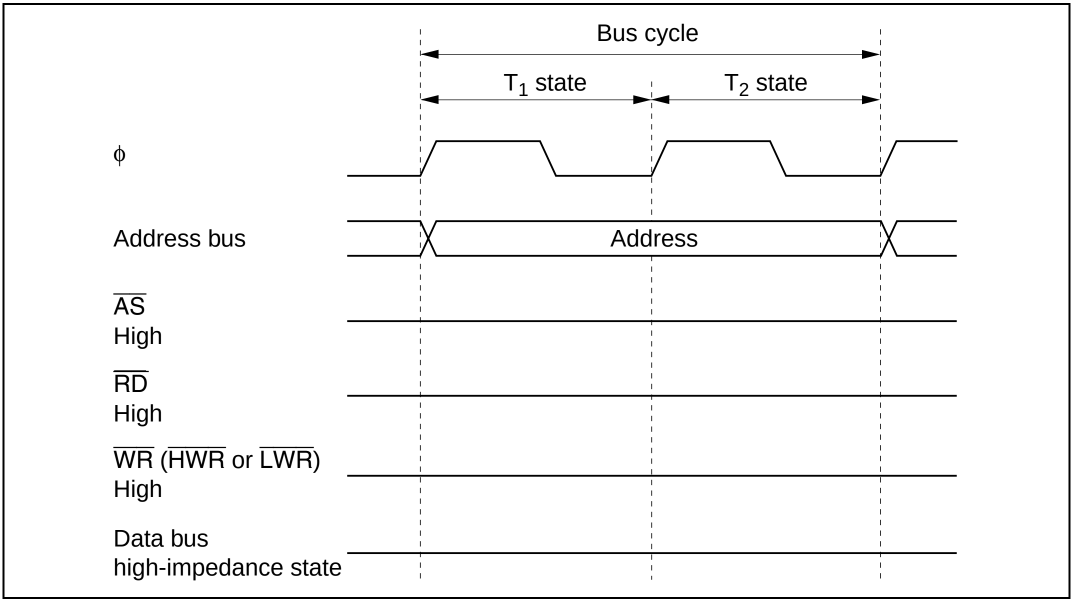


Figure 4.2   Pin States during On-Chip Memory Access

4.3 On-Chip Supporting Modules

The on-chip supporting modules are accessed in three states. The data bus is 8 bits or 16 bits wide. Figure 4.3 shows the access timing for the on-chip supporting modules. Figure 4.4 shows the pin states.

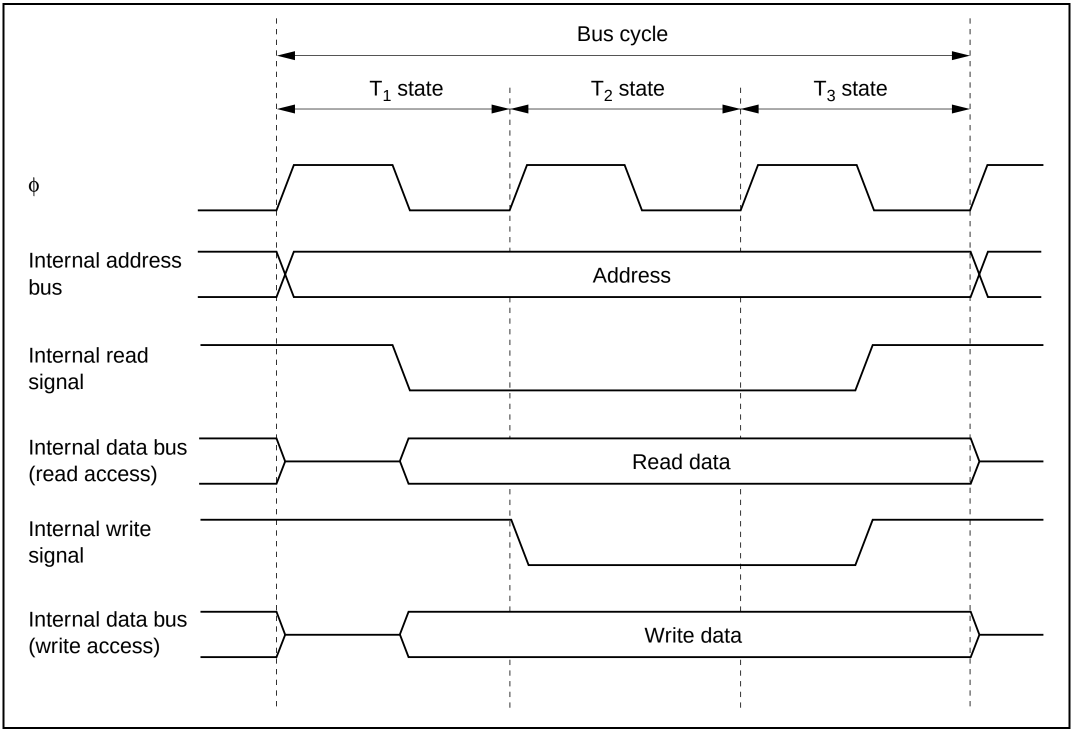


Figure 4.3 On-Chip Supporting Module Access Cycle

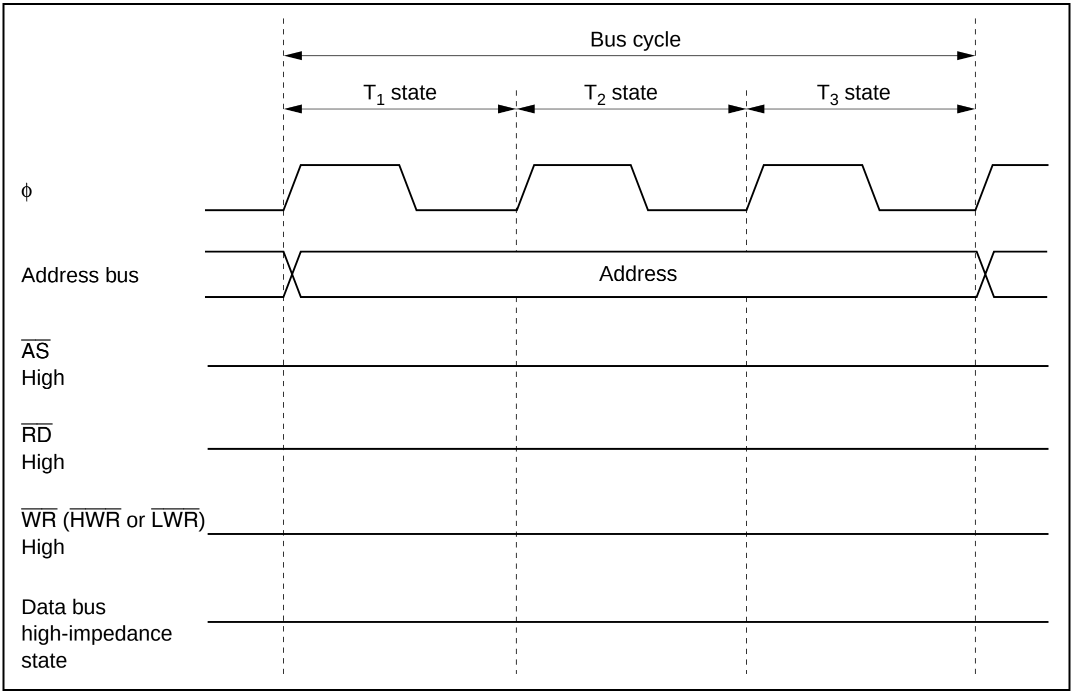


Figure 4.4 Pin States during On-Chip Supporting Module Access

4.4 External Data Bus

The external data bus is accessed with 8-bit or 16-bit bus width in two or three states. Figure 4.5 shows the read timing for two-state or three-state access. Figure 4.6 shows the write timing for two-state or three-state access. In three-state access, wait states can be inserted by the wait-state controller or other means. For further details refer to the relevant microcontroller hardware manual.

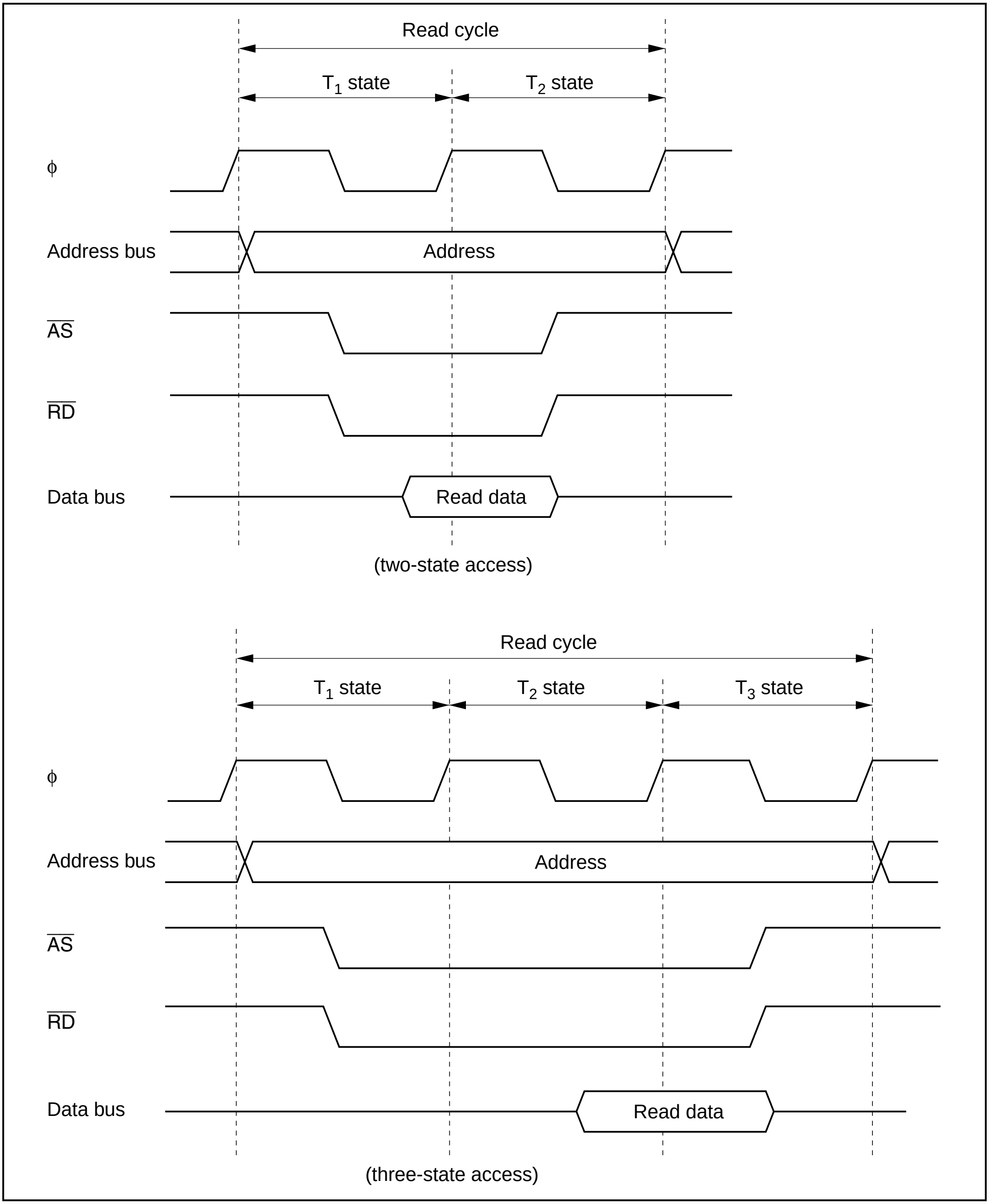
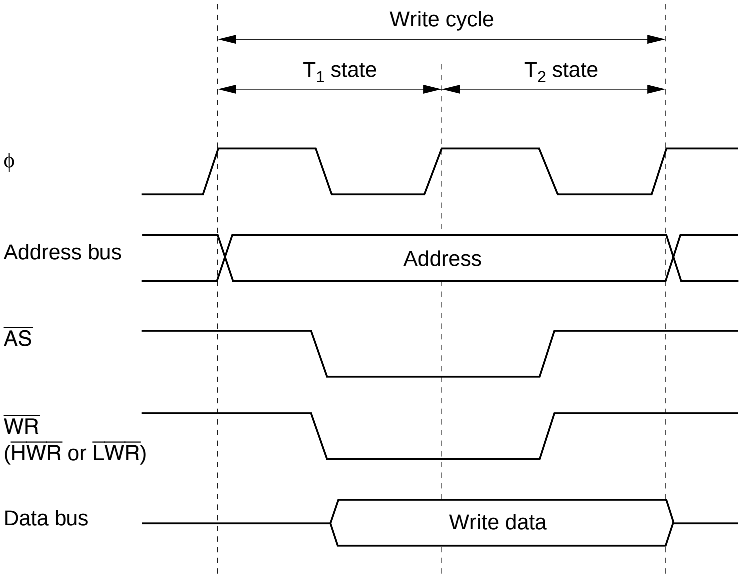
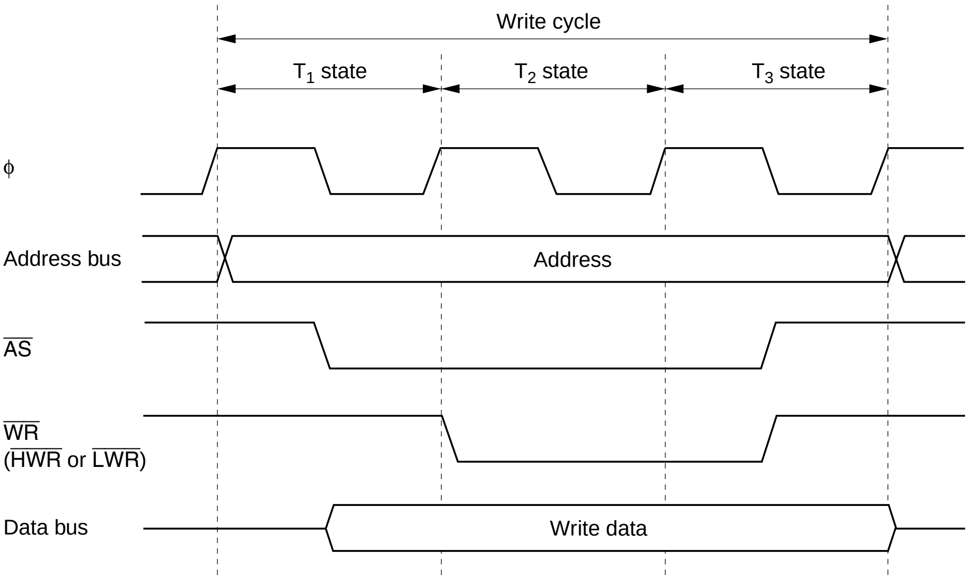


Figure 4.5 External Device Access Timing (1) Read Timing



(a) Two-state access



(b) Three-state access

Figure 4.6 External Device Access Timing (2) Write Timing

Renesas 16-Bit Single-Chip Microcomputer
Software Manual
H8/300H Series

Publication Date:	1st Edition, August 1993
	Rev.3.00, December 13, 2004
Published by:	Sales Strategic Planning Div.
	Renesas Technology Corp.
Edited by:	Technical Documentation & Information Department
	Renesas Kodaira Semiconductor Co., Ltd.

Renesas Technology Corp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan

RENESAS SALES OFFICES

http://www.renesas.com

Refer to "**http://www.renesas.com/en/network**" for the latest and detailed information.

Renesas Technology America, Inc.
450 Holger Way, San Jose, CA 95134-1368, U.S.A
Tel: <1> (408) 382-7500, Fax: <1> (408) 382-7501

Renesas Technology Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.
Tel: <44> (1628) 585-100, Fax: <44> (1628) 585-900

Renesas Technology Hong Kong Ltd.
7th Floor, North Tower, World Finance Centre, Harbour City, 1 Canton Road, Tsimshatsui, Kowloon, Hong Kong
Tel: <852> 2265-6688, Fax: <852> 2730-6071

Renesas Technology Taiwan Co., Ltd.
10th Floor, No.99, Fushing North Road, Taipei, Taiwan
Tel: <886> (2) 2715-2888, Fax: <886> (2) 2713-2999

Renesas Technology (Shanghai) Co., Ltd.
Unit2607 Ruijing Building, No.205 Maoming Road (S), Shanghai 200020, China
Tel: <86> (21) 6472-1001, Fax: <86> (21) 6415-2952

Renesas Technology Singapore Pte. Ltd.
1 Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: <65> 6213-0200, Fax: <65> 6278-8001

Colophon 2.0

H8/300H Series
Software Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ09B0213-0300